Tree Based Symmetric Key Broadcast Encryption

Sanjay Bhattacherjee and Palash Sarkar
Applied Statistics Unit
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
sanjay.bhattacherjee@gmail.com and palash@isical.ac.in

Abstract

The most influential broadcast encryption (BE) scheme till date was introduced in 2001 by Naor, Naor and Lotspiech (NNL) and is based on binary trees. This paper generalizes the ideas of NNL to obtain BE schemes based on k-ary trees for any $k \geq 2$. The treatment is uniform across all k and essentially provides a single scheme which is parameterized by the arity of the underlying tree. We perform an extensive analysis of the header length and user storage of the scheme. It is shown that for a k-ary tree with n users out of which r are revoked, the maximum header length is $\min(2r-1, n-r, \lceil n/k \rceil)$. An expression for the expected header length is obtained and it is shown that the expression can be evaluated in $O(r \log n)$ time. Experimental results indicate that for values of r one would expect in applications such as pay TV systems, the average header length decreases as k increases. The number of keys to be stored by any user is shown to be at most $(\chi_k - 2)\ell_0(\ell_0+1)/2$, where $\ell_0 = \lceil \log_k n \rceil$ and χ_k is the number of cyclotomic cosets modulo $2^k - 1$. In particular, when the number of users is more than 1024, we prove that the user storage required for $k = 3$ is less than that of $k = 2$. For higher values of k, the user storage is greater than that for binary trees. The option of choosing the value of k provides a designer of a BE system with a wider range of trade-offs between average header length and user storage. The effect of layering on the k-ary tree SD scheme is also explored.

Keyword: Broadcast encryption; subset difference; trees; general arity; probabilistic analysis; header length; transmission overhead; cyclotomic cosets; layering

1 Introduction

Broadcast Encryption (BE) deals with the problem of broadcasting encrypted data. For each transmission (or session), there is a set of *privileged* users who should be able to decrypt the data and a set of *revoked* users who should not be able to do so. In symmetric key BE, there is a center which initially distributes keys to all the users and also broadcasts the encrypted data in each session. In each session, the data to be broadcast is encrypted with a random session key using a symmetric key encryption algorithm. This session key is further encrypted using other keys and the encryptions of the session key are sent as the *header* with the encrypted body. The number of times the session key is encrypted for each session is called the *header length*. Any privileged user will be able to use its secret information to correctly decrypt the session key from the header and hence the message sent in the session. A *fully resilient* scheme ensures that an adversary with the secret information of all the revoked users cannot decrypt the broadcast correctly. Two important efficiency parameters for a BE scheme are the header length; and the user storage which is the amount of secret information that each user has to store.

The most popular symmetric key BE scheme was proposed in 2001 by Naor, Naor and Lotspiech (NNL) and was called the Subset Difference (SD) Scheme [NNL01, NNL02]. The NNL-SD scheme assumes the number of users to be a power of two and these users are associated with the leaves of a *full binary tree* $\mathcal{T}^0$. This scheme

has a storage requirement of $O(\log^2 n)$ and a worst case header length of $2r-1$, where n is the total number of users and r is the number of revoked users.

The idea of layering introduced by Halevy and Shamir [HS02] reduced the user storage of the NNL-SD scheme from $O(\log^2 n)$ to $O(\log^{3/2} n)$ at the cost of increased communication overhead. In [BS14], this layering strategy was generalized to obtain different trade-offs between the user storage and communication overhead. An important optimization was the storage minimal layering that guaranteed minimal storage requirement that could be achieved using layering for a given n.

1.1 Our Contributions

In this work, we extend the ideas of NNL to k-ary trees for any $k \geq 2$. Our treatment is general and unified, i.e., the same approach works for all values of k. Suppose n is a power of k, i.e., $n = k^{\ell_0}$ for some $\ell_0 \geq 1$ and consider the users to be the leaf nodes of a full k-ary tree of height ℓ_0. Let $j_1, \ldots, j_c$, $1 \leq c \leq k$, be a set of sibling nodes in this tree and i is an ancestor of these nodes. Consider the set S of leaf nodes in the subtree formed by taking away the subtrees rooted at $j_1, \ldots, j_c$ from the subtree rooted at i. So, the set S is formed as a subset difference of two sets of users. Subsets of users arising in this manner are called SD sets. The identification of the SD sets is a key aspect of obtaining the k-ary tree scheme. This idea extends the idea of SD sets introduced for binary trees in [NNL01, NNL02].

An intuition behind considering k-ary trees with $k > 2$ is that the number of SD sets grows with increasing k and so the header length may come down at the cost of increasing the user storage. This, however, does not turn out to be entirely true. Working out the details of the scheme and the resulting analysis shows up a rich complexity of behavior which is not apparent at the outset. We provide an extensive analysis of the scheme covering the following points.

Cover Generation Algorithm A single cover generation algorithm which works for all k is developed. This is an intuitively simple algorithm which uses just an array as the underlying data structure. Specializing this algorithm for $k = 2$ yields the cover finding algorithm given in [NNL01, NNL02]. The description of the algorithm turns out to be considerably simpler that of [NNL01, NNL02].

Traitor Tracing The NNL paper [NNL01, NNL02] provides a mechanism for tracing traitors. With some modification, this idea also fits the k-ary BE scheme. It turns out that compared to binary trees, for $k \geq 3$, tracing traitors can be done with lesser number of queries.

Header Length For k-ary trees with n users, the maximum header length of a transmission with r revoked users is shown to be $\min(2r-1, n-r, \lceil n/k \rceil)$. Somewhat surprisingly, the first component, i.e., $2r-1$ is not affected by k. We show that the bound of $2r-1$ is indeed achieved for values of k greater than 2. Average case analysis of the header length is done under the assumption that the revoked set of users is distributed uniformly among the set of all users. With this assumption, we derive an expression for the expected header length. The method is to compute the probability that any internal node generates a subset in the header. Summing over all these probabilities provide the expected header length. The expression for the expected header length can be computed in $O(r \log n)$ time and $O(1)$ space. We have implemented the algorithm to compute the expected header length and provide representative values to show the average header lengths for different values of k.

User Storage During the initiation of the scheme, the center provides each user with sufficient information so that it is able to generate any key corresponding to an SD set of which it is a member. This information is measured in terms of the number of m-bit seeds that are required to be stored by any user. Here m is the size of the key of the underlying symmetric cipher. The work of NNL provides a clever way to use a pseudo-random

generator so that user storage consists of $1 + \lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)/2$ seeds. The direct combination of this idea with the SD sets of a k-ary tree makes the user storage to be $1 + (2^{k-1} - 1)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds. We show that a modification based on the use of cyclotomic cosets modulo $2^k - 1$ reduces the user storage to $1 + (\chi_k - 2)\lceil \log_k n \rceil (\lceil \log_k n \rceil + 1)/2$ seeds, where χ_k is the number of cyclotomic cosets modulo $2^k - 1$.

Tackling Arbitrary Number Of Users When n is not a power of k, we show that a complete k-ary tree structure can be used to construct the BE scheme. This is an analogue of complete binary trees used in data structures. Average header length analysis of such schemes is performed using simulation studies.

Layering The idea of layering is extended for the k-ary tree generalization of the SD scheme. The choice of the layering strategy determines the user storage of the layered version of the scheme. A dynamic programming algorithm is proposed to compute the layering strategies for which the user storage is minimum. This generalizes the algorithm for $k = 2$ which was given in [BS14].

Simulation Study Of The Header Length We perform a simulation study of the average header length for $n = 10^x$ $(x = 3, \ldots, 8)$ users and for $k = 2, \ldots, 8$. Experimental results indicate that there is a cut-off value δ_k such that for $r/n > \delta_k$, the average header length of the k-ary scheme is lesser than that of the binary tree based scheme. Further, the value of δ_k decreases as k increases. This suggests that by increasing k, it is possible to reduce the header length for lower values of r. This can be important for applications such as Pay-TV systems. The trade-off is a one-time moderate increase in user storage.

In applications like [AAC] where the storage for the header is fixed in the standard, there is a threshold for the maximum number of users that the system can accommodate. With decrease in the average header length, the system can accommodate more number of revoked users on an average.

1.2 Previous And Related Works

Broadcast Encryption was introduced by Berkovitz and his scheme was based on "k out of n" secret sharing [Ber91]. However, this scheme was "one-time" and the keys had to be updated after every use. BE was first formally studied and the term Broadcast Encryption was coined by Fiat and Naor in [FN93]. The idea of resiliency of a scheme against colluding users was also formally introduced in [FN93]. They proposed a number of schemes that provided trade-offs between the user storage and the transmission length.

Broadcast Encryption for stateless receivers was first considered explicitly by Naor, Naor and Lotspiech in [NNL01, NNL02]. They defined a generic framework, called Subset-Cover framework, encapsulating several previously proposed revocation methods. The algorithms of the Subset-Cover framework are based on the principle of covering all non-revoked users by disjoint subsets from a predefined collection. They proposed two tree-based schemes namely the Complete Subtree (NNL-CS) scheme and the Subset Difference (NNL-SD) scheme.

Halevy and Shamir [HS02] reduced the user storage in the NNL-SD scheme by dividing the levels of the underlying tree of the SD scheme into layers. The NNL-SD [NNL01, NNL02] and the HS-LSD [HS02] schemes assumed the number of users to be a power of two. In [BS14] the idea of layering was further explored to obtain various layering strategies offering different trade-offs. One out of these optimizations was minimizing the user storage.

Two public key BE schemes were proposed in [Asa02] that assigned keys to subsets following the Complete Subtree (CS) method of [NNL01, NNL02]. While the CS method assumed an underlying binary tree, the schemes in [Asa02] were allowed to have arity greater than or equal to two. The paper does not discuss how to extend the Subset Difference technique of [NNL01, NNL02] for trees of arity greater than two.

The Subset Difference technique of [NNL01, NNL02] was extended for ternary trees in [FKTS08]. The key assignment technique of [FKTS08] however could not be extended to higher arities. To quote from their paper (page 236 of the WISA 2008 proceedings):

"However, in a general a-array tree with $a \geq 4$, there exists sets of nodes that are inconsecutive ... Our hash chain approach fails with regard to these inconsecutive points. Thus, the construction of a coalition resistant a-array SD method with reasonable communication, computation, and storage overhead is an open issue."

For a given upper bound on the user storage of any BE scheme, a lower bound on the header length was shown in [LS98]. Stateless and low-state BE in low-memory devices was considered in [GST04]. One-way chain based BE schemes were proposed in [JHC+05] where the header length was brought down below r for the first time, while the user storage requirement was quite large. The NNL-SD scheme can also be viewed as a special case of this scheme.

There have been several works on the analysis of the tree based schemes. The analysis of the expected header length for the NNL-CS, the NNL-SD and the HS-LSD schemes was done in [PB06]. This analysis was continued in [EOPR08] which showed that the standard deviations for these schemes are small compared to the means as the number of users gets large. Therefore, the mean number is a good estimate of the number of necessary encryptions used by these schemes. A detailed analysis of expected header lengths and algorithms to compute them have been obtained in [BS13]. Lower bounds on the header length of most subset cover algorithms for different types of r were found in [AK08]. Detailed worst case header length analysis for the NNL-CS and NNL-SD schemes of [NNL01, NNL02] have been done in [MMW09].

2 Subset Cover Framework

The Subset-Cover framework for broadcast encryption was proposed in
[NNL01, NNL02]. Almost every known symmetric key broadcast encryption scheme falls under this framework. It has three phases namely, *initiation*, *encryption* and *decryption*. In the initiation phase, a collection $\mathcal{S}$ of subsets of $\mathcal{N}$ is created. Each subset in $\mathcal{S}$ is assigned a secret key. A user gets the secret information which enables it to compute the secret keys of all subsets S of users such that it is a member of S and $S \in \mathcal{S}$. Once this initiation phase is over, the system is ready for secured broadcasting. The broadcast messages are sent to the users in blocks. Each block goes with a new session. For each session, the center knows the set of revoked users $\mathcal{R}$. It finds the set of subsets $S_c = \{S_{i_1}, S_{i_2}, \ldots, S_{i_h}\} \subset \mathcal{S}$ from the collection $\mathcal{S}$ such that each privileged user is in some subset in S_c. In other words,

$$\mathcal{N} \setminus \mathcal{R} = \bigcup_{i \in \{i_1, \ldots, i_h\}} S_i.$$

This set of subsets S_c is called the *subset cover* and the algorithm to find S_c is called the *cover generation or cover finding algorithm*. A message block is encrypted with a random *session key* K chosen anew for each session and forms the body of the message. This session key is further encrypted for each subset S_i in S_c, using its key. These encryptions of the session key accompany the message body as the header. The number of subsets in the cover $h = |S_c|$ is called the header length. For decryption, a privileged user first decrypts the session key from the header, using which it decrypts the message block.

2.1 The (Binary Tree) Subset Difference Scheme

The Naor-Naor-Lotspiech Subset Difference (NNL-SD) scheme [NNL01, NNL02] is the most popular BE scheme and falls under the Subset-Cover framework. The NNL-SD scheme assumes the number of users to be a power of two. During initiation, the users are associated with the leaves of a *full binary tree* $\mathcal{T}^0$. A subtree of $\mathcal{T}^0$ rooted at node i in the tree is denoted as $\mathcal{T}^i$. A subtree $\mathcal{T}^i$ also identifies the set of users at its leaves. The subsets in the collection $\mathcal{S}$ are of the form $S_{i,j}$ where $S_{i,j}$ consist of the leaf nodes of the subgraph $\mathcal{T}^i \setminus \mathcal{T}^j$ such that j ($\neq i$) is a node in the subtree rooted at node i of $\mathcal{T}^0$.

Keys are associated to each subset $S_{i,j} \in \mathcal{S}$ as follows. Every internal (non-leaf) node i in $\mathcal{T}^0$ is assigned a uniform random seed L_i. All descendant nodes j in the subtree $\mathcal{T}^i$ are associated with seeds derived from L_i in the following manner. A cryptographic pseudo-random generator $G : \{0,1\}^k \to \{0,1\}^{3k}$ is chosen. The output of G is thrice the length of its input. The three equal portions of the output are denoted as $G(seed) = G_L(seed)||G_M(seed)||G_R(seed)$. Let a node i have some (random or derived) $seed_i$. The left child of i gets the derived seed $G_L(seed_i)$ and the right child of i gets the derived seed $G_R(seed_i)$. The derived seed for a node j from the uniform random seed L_i of an ancestor node i is denoted as $L_{i,j}$. The key for the subset $S_{i,j}$ is $G_M(L_{i,j})$.

A user receives the secret information from which it can derive the secret keys of all subsets to which it belongs. It is to be noted that if a user belongs to a subset $S_{i,j}$, i is an ancestor of the leaf corresponding to the user and j (being in the subtree $\mathcal{T}^i$), is not an ancestor of that leaf. Let i be an ancestor of the leaf node associated with the user. Let j_1 be a node directly attached to the path joining the user leaf and the node i. The seed derived from L_i by the node j_1 is L_{i,j_1}. The user receives this derived seed L_{i,j_1} as part of its secret information. It can be seen that using L_{i,j_1}, the user can derive the key for any subset $S_{i,j}$ where j is in the subtree rooted at node j_1. Similarly, the user receives the seeds derived from L_i for all nodes directly attached to the path joining the leaf of the user and the ancestor i. From the derived seeds that the user gets, it can derive the keys for all subsets $S_{i,j}$ to which it belongs.

A cover finding algorithm is described in [NNL01, NNL02] which allows the center to find a collection of subsets which covers all the privileged users. We do not describe this here, since, later we provide a cover finding algorithm for k-ary trees which has the binary tree cover finding algorithm as a special case.

3 The k-ary Tree Subset Difference Scheme

The description of the scheme is given in two parts – initiation and the cover generation algorithm.

3.1 Initiation

Fix the arity of the underlying tree to be a positive integer $k \geq 2$ and let the number of users n to be a power of k, say $n = k^{\ell_0}$. (Later, we describe how to handle the case when n is not a power of k.) Let $\mathcal{T}^0$ be a full k-ary tree having $n = k^{\ell_0}$ leaf nodes. There are $\ell_0 + 1$ levels in $\mathcal{T}^0$. The root node is considered to be at level ℓ_0 while the leaf nodes are considered to be at level 0. The total number of nodes in $\mathcal{T}^0$ is

$$1 + k + k^2 + \ldots + k^{\ell_0} = \frac{nk - 1}{k - 1}.$$

The users are assumed to be at the leaf nodes of $\mathcal{T}^0$. So, the set $\mathcal{N}$ of all users consists of the leaf nodes of $\mathcal{T}^0$.

Numbering Of The Nodes In $\mathcal{T}^0$ The nodes in $\mathcal{T}^0$ are numbered as follows: the root is numbered 0; the k children of an internal node i are numbered from left to right as $ki + 1$, $ki + 2$, $\ldots$, $ki + k$. The nodes in $\mathcal{T}^0$ are identified by their numbers. So, the parent of any node i is $\lfloor \frac{i-1}{k} \rfloor$. For a node i, we denote by $\mathcal{T}^i$ the subtree of $\mathcal{T}^0$ rooted at i.

The Collection $\mathcal{S}$ Let i be an internal node and suppose J is a set of nodes in $\mathcal{T}^i$ which have a common parent (and so the nodes in J are siblings) such that $1 \leq |J| < k$. Let $S_{i,J}$ be the set of leaf nodes in the subgraph

$$\mathcal{T}^i \setminus \bigcup_{j \in J} \mathcal{T}^j.$$

$S_{i,J}$ is a subset of the leaf nodes of $\mathcal{T}^0$ and so a subset of the set of all users $\mathcal{N}$. Define $\mathcal{S}$ to be the collection of all possible $S_{i,J}$ and also $\mathcal{N}$. Keys are assigned only to the subsets of users in $\mathcal{S}$ and to no other subsets of $\mathcal{N}$.

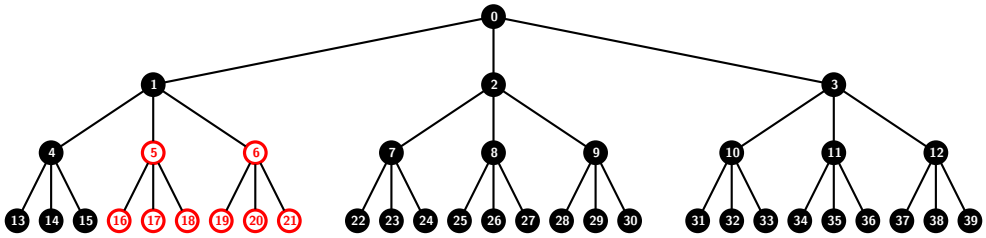

Figure 1: The k-ary tree $\mathcal{T}^0$ with $k = 3$ and $n = 27$ users. The subset $S_{0,\{5,6\}}$ contains all users (leaves) in the subtree $\mathcal{T}^0$ but not in $\mathcal{T}^5$ or $\mathcal{T}^6$. Hence, $S_{0,\{5,6\}} = \{13, 14, 15, 22, 23, \ldots, 39\}$.

Number Of Subsets In The Collection $\mathcal{S}$ We count the number of subsets $S_{i,J}$ in $\mathcal{S}$. Fix an internal node i and suppose it is at level ℓ. Let us now consider the number of subsets of nodes J such that $S_{i,J}$ is in $\mathcal{S}$. There are two conditions on J: all nodes in J have the same parent and $1 \leq |J| \leq k-1$. The common parent of the nodes in J is an internal node in $\mathcal{T}^i$. So an internal node in $\mathcal{T}^i$ gives rise to $2^k - 2$ possible subsets of nodes J. The number of internal nodes in $\mathcal{T}^i$ is $1 + k + k^2 + \ldots + k^{\ell-1} = (k^\ell - 1)/(k-1)$. So for a fixed node i at level ℓ, there are a total of $(2^k - 2)((k^\ell - 1)/(k-1))$ possible subsets $S_{i,J}$.

In $\mathcal{T}^0$, there are $k^{\ell_0 - \ell}$ internal nodes at level ℓ. Therefore, the number of subsets generated by all the nodes at level ℓ is $(k^\ell - 1)/(k-1) \times (2^k - 2) \times k^{\ell_0 - \ell}$. Summing this by varying ℓ from 0 to ℓ_0 we get the total number of subsets $S_{i,J}$ in $\mathcal{S}$. Additionally, we have to count the set $\mathcal{N}$ of all users. Hence, the total number of subsets in $\mathcal{S}$ is

$$\begin{aligned} |\mathcal{S}| = \; & 1 + (2^k - 2) \textstyle\sum_{\ell=1}^{\ell_0} \frac{k^\ell - 1}{k-1} (k^{\ell_0 - \ell}) = \; 1 + \frac{2^k - 2}{k-1} \sum_{\ell=0}^{\ell_0 - 1} (n - k^\ell) \\ = \; & 1 + \tfrac{2^k - 2}{k-1} \left(n\ell_0 + \tfrac{n-1}{k-1} \right). \end{aligned} \tag{1}$$

For $n = 16$, and $k = 2$, the number of subsets in $\mathcal{S}$ is 159. For $n = 16$, and $k = 4$, the number of subsets in $\mathcal{S}$ is 323. We observe that for a fixed n, the number of subsets in the collection increases with increasing k. Intuitively, it seems that increasing the number of subsets in $\mathcal{S}$ by increasing k should decrease the header length. This, however, is not always true. Later, we make a detailed analysis of both the maximum and the average header length of the scheme.

Key Assignment To Subsets In $\mathcal{S}$ Given an m-bit string, we need to obtain $2^k - 1$ m-bit strings. This is achieved as follows:

Let $G : \{0, \ldots, 2^k - 2\} \times \{0,1\}^m \to \{0,1\}^m$ be a cryptographic hash function. Define $G_\sigma(seed) \triangleq G(\sigma, seed)$. This defines $G_\sigma(seed)$ for an m-bit string $seed$ and $0 \leq \sigma \leq 2^k - 2$. One advantage of this method is that given σ it allows directly "jumping" to a particular $G_\sigma(seed)$. We note though that the description of the key assignment method given below does not depend on the particular manner in which G_σ has been defined.

The key assigned to a subset $S_{i,J}$ is defined indirectly. The procedure is described as follows.

1. Every internal node i is assigned an independent and uniform random seed L_i.
2. Every node $j \neq i$ in the subtree $\mathcal{T}^i$ is assigned a seed $L_{i,\{j\}}$ derived from L_i using G in the following manner.
   (a) Suppose j is an immediate child of i and write j as $j = ki + s + 1$ for some $0 \leq s \leq k-1$. Define $L_{i,\{j\}} = G_{2^s}(L_i)$.

(b) If j is not an immediate child of i, then let $i = t_0, \ldots, t_p = j$ be a sequence of nodes from i to j. Let $t_q = kt_{q-1} + s_q + 1$ where $0 \leq s_q \leq k-1$ for $0 \leq q \leq p$. Define $L_{i,\{j\}} = G_{2^{s_p}}(G_{2^{s_{p-1}}}(\cdots G_{2^{s_1}}(L_i)))$.

3. Let j (possibly equal to i) be an internal node in $\mathcal{T}^i$ and let $J \subset \{kj+1, kj+2, \ldots, kj+k\}$ with $2 \leq |J| \leq k-1$. The previous step has already defined the seed $L_{i,\{j\}}$. Let s be the unique integer in $\{0, \ldots, 2^k - 2\}$ such that the k-bit binary representation of s encodes J, i.e., the bth bit of this binary representation is 1 if and only if $kj + b$ is in J. Define $L_{i,J} = G_s(L_{i,\{j\}})$.

4. For each possible subset $S_{i,J}$, the above procedure defines the seed $L_{i,J}$. The key assigned to the subset $S_{i,J}$ is $G_0(L_{i,J})$.

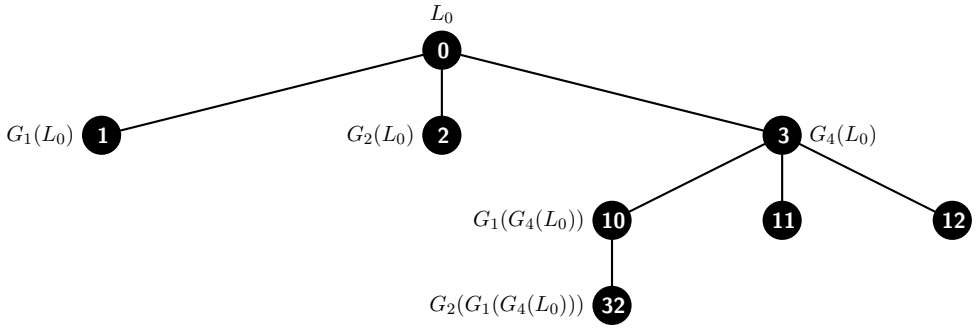

Figure 2: Seeds derived by node 32 and its ancestors 3 and 10 from L_0. $L_{0,\{32\}} = G_2(G_1(G_4(L_0)))$.

To illustrate the assignment of seeds, let us consider the tree $\mathcal{T}^0$ with $k = 3$ and $n = 27$ users as shown in Figure 1. The internal nodes $0, \ldots, 12$ get uniform random seeds $L_0, \ldots, L_{12}$ respectively. The seeds derived from L_0 by nodes at levels 2 and 1 are as follows:

For level 2:
$L_{0,\{1\}} = G_1(L_0)$, $L_{0,\{2\}} = G_2(L_0)$, $L_{0,\{3\}} = G_4(L_0)$,
$L_{0,\{1,2\}} = G_3(L_0)$, $L_{0,\{2,3\}} = G_6(L_0)$, $L_{0,\{1,3\}} = G_5(L_0)$.
For level 1:
$L_{0,\{4\}} = G_1(G_1(L_0))$, $L_{0,\{5\}} = G_2(G_1(L_0))$, $L_{0,\{6\}} = G_4(G_1(L_0))$,
$L_{0,\{4,5\}} = G_3(G_1(L_0))$, $L_{0,\{5,6\}} = G_6(G_1(L_0))$, $L_{0,\{4,6\}} = G_5(G_1(L_0))$,
$L_{0,\{7\}} = G_1(G_2(L_0))$, $L_{0,\{8\}} = G_2(G_2(L_0))$, $L_{0,\{9\}} = G_4(G_2(L_0))$,
$L_{0,\{7,8\}} = G_3(G_2(L_0))$, $L_{0,\{8,9\}} = G_6(G_2(L_0))$, $L_{0,\{7,9\}} = G_5(G_2(L_0))$,
$L_{0,\{10\}} = G_1(G_4(L_0))$, $L_{0,\{11\}} = G_2(G_4(L_0))$, $L_{0,\{12\}} = G_4(G_4(L_0))$,
$L_{0,\{10,11\}} = G_3(G_4(L_0))$, $L_{0,\{11,12\}} = G_6(G_4(L_0))$, $L_{0,\{10,12\}} = G_5(G_4(L_0))$.

Similarly, the seeds derived from L_0 for subtrees at level 0 and their combinations, can be determined.

Figure 2 shows how node 32 of Figure 1 gets its derived seed from the uniform random seed L_0. There will be seeds derived from every such L_i.

Storage Per User During initiation, a user will receive information (a set of derived seeds), from which it can derive the keys of all subsets it belongs to and no more. A user is associated to a leaf node in $\mathcal{T}^0$. The SD subsets $S_{i,J}$ the user will belong to, will be rooted at some ancestor node i of that leaf. However, none of the nodes in J will be an ancestor of that leaf. Thus, a user belongs to all subsets $S_{i,\{j_1,\ldots,j_s\}}$ for which

- i is an ancestor of the user leaf, and
- none of the nodes $j_1, \ldots, j_s$ are on the path joining the root node and the user leaf.

A user has to receive seeds such that it can generate the keys of all such subsets. We have already seen how keys for subsets are derived from seeds assigned to nodes in $\mathcal{T}^0$. Out of these seeds, a user gets the derived seeds from which it can generate the keys of subsets to which it belongs and no more.

The general strategy for assignment of seeds to users is as follows. Let us consider the path joining the user leaf and the root node in $\mathcal{T}^0$. Let i be a node on this path and hence an ancestor of that user. The key for a subset $S_{i,J}$ to which the user belongs will be derived from L_i. None of the nodes in J are on the path joining the user leaf and i (a part of the path with the root). Hence, the nodes in J are siblings that are either directly attached to this path or are in a subtree attached to this path. *The user gets the seeds derived from L_i of all nodes and their combinations that are directly attached with (or "falling-off" from) this path.* Using these seeds and G, the user can derive the keys of every subset $S_{i,J}$ to which it belongs and no more.

To illustrate the assignment of seeds to the users, let us again consider the tree in Figure 1. As an example, we look at the information that has to be given to the user at leaf 13. For that, we first identify the subsets to which the user at 13 belongs. The user at leaf 13 has three ancestor nodes $4, 1$ and 0. Hence, it belongs to subsets of the form $S_{0,J}$, $S_{1,J}$ and $S_{4,J}$ where nodes in the respective subsets J are not ancestors of the leaf 13. If the user at leaf 13 gets the derived seed $L_{0,\{2\}}$, it can derive using G, the key for any subset $S_{0,J}$ where nodes in J are in the subtree rooted at node 2. Similarly, with the derived seed $L_{0,\{3\}}$, the user can derive the key for any subset $S_{0,J}$ where nodes in J are in the subtree rooted at node 3. Additionally, it needs the key for the subset $S_{0,\{2,3\}}$. We know that, if the user at leaf 13 gets the derived seeds $L_{i,J}$ for every ancestor node i and the set J has a node (or a combination of nodes) directly attached to the path joining the leaf 13 to the root node 0, it can derive the key for any subset it belongs to.

Using this strategy, the user at leaf node 13 gets the seeds for $S_{0,J}$ for the following J: $\{2\}$, $\{3\}$, $\{2,3\}$, $\{5\}$, $\{6\}$, $\{5,6\}$, $\{14\}$, $\{15\}$, $\{14,15\}$. It gets the seeds for $S_{1,J}$ for the following J: $\{5\}$, $\{6\}$, $\{5,6\}$, $\{14\}$, $\{15\}$, $\{14,15\}$. It gets the seeds for $S_{4,J}$ for the following J: $\{14\}$, $\{15\}$, $\{14,15\}$. Hence, the user at leaf node 13 gets the following seeds:

derived from L_0:
$G_2(L_0)$, $G_4(L_0)$, $G_6(L_0)$,
$G_2(G_1(L_0))$, $G_4(G_1(L_0))$, $G_6(G_1(L_0))$,
$G_2(G_1(G_1(L_0)))$, $G_4(G_1(G_1(L_0)))$, $G_6(G_1(G_1(L_0)))$.
derived from L_1:
$G_2(L_1)$, $G_4(L_1)$, $G_6(L_1)$,
$G_2(G_1(L_1))$, $G_4(G_1(L_1))$, $G_6(G_1(L_1))$.
derived from L_4:
$G_2(L_4)$, $G_4(L_4)$, $G_6(L_4)$.

Next we compute the number of seeds that the user will have to store. The number of seeds derived from seed L_i of an ancestor node i at level ℓ, will be $2^{k-1}-1$ for each level below ℓ. Thus, the total number of derived seeds due to node i will be $(2^{k-1}-1)\ell$. Since there are ℓ_0 such ancestor nodes of the user at each level $1, \ldots, \ell_0$, the total number of seeds to be stored by the user will be

$$1 + (2^{k-1}-1)\sum_{\ell=1}^{\ell_0} \ell = 1 + \frac{\ell_0(\ell_0+1)}{2}(2^{k-1}-1). \tag{2}$$

The addition of 1 in the above expression is due to the key that is assigned to the set $\mathcal{N}$ of all users. Each user will be required to store this key to decrypt a message that is broadcast to all the users, i.e., when there are

no revoked users. The factor $(2^{k-1}-1)$ in (2) can be reduced using a modified method of distributing secret information to the users. We describe how to do this in Section 6.

3.2 Cover Finding Algorithm

Once the initiation is over and users have been given their secret information, the center can start broadcasting encrypted messages to the set of privileged users. If there is no revoked user, the only set for which the messages are encrypted is the set $\mathcal{N}$ of all users. Otherwise, for a given set of revoked users, the center finds the subset cover using the iterative algorithm outlined below. The subset cover contains subsets of the form $S_{i,J}$ where all nodes in J are siblings and hence are at the same level.

The algorithm runs on a list $\mathcal{L}$ of nodes in $\mathcal{T}^0$ that lie on the paths joining revoked leaf nodes to the root node. To start with, the list $\mathcal{L}$ consists of all revoked leaves from left to right in $\mathcal{T}^0$. In the course of the algorithm, $\mathcal{L}$ is appended with all nodes on the paths joining the revoked leaves to the root. This is done as follows. Once $\mathcal{L}$ is populated with the revoked leaves, the algorithm runs iteratively from left to right on $\mathcal{L}$. In iteration t, it considers the t^{th} node j from the left in $\mathcal{L}$. If j is not the root, the parent i of j is appended to $\mathcal{L}$, if it is not already present there. Hence $\mathcal{L}$ keeps growing on the right, with nodes at higher levels on the tree (up to the root) getting added to its right end. The root node eventually gets appended to $\mathcal{L}$. The algorithm terminates after working on the root node.

For each node j in $\mathcal{L}$, a summary of requisite information about the subtree $\mathcal{T}^j$ is maintained in $\mathcal{L}$ along with the node j. Each node j in $\mathcal{L}$ has an associated set $\mathsf{SDnodes}[j]$. The set $\mathsf{SDnodes}[j]$ contains roots of all subtrees that will be subtracted from $\mathcal{T}^j$, in case an SD subset is generated from node j. The cover finding algorithm ensures that all nodes in $\mathsf{SDnodes}[j]$ for any node j are at the same level in $\mathcal{T}^0$. For each leaf node j in the initial list $\mathcal{L}$, $\mathsf{SDnodes}[j] = \{j\}$. If a node j gives rise to a subset $S_{j,J}$ in the algorithm, then $J = \mathsf{SDnodes}[j]$. In the course of the algorithm, each such node j from which a subset $S_{j,\mathsf{SDnodes}[j]}$ should be generated, have to be identified. To that end, each node in $\mathcal{L}$ gets marked as "intermediate" or "covered" depending upon its position in the tree. Every iteration of the algorithm works on a particular node and based on the mark of that node and its siblings, subsets for the cover may or may not be generated. Let the t^{th} node from the left in $\mathcal{L}$ be denoted by $\mathcal{L}[t]$. The node $\mathcal{L}[t]$ is processed in the t^{th} iteration.

The Algorithm Takes as input the set $\mathcal{R}$ of revoked users and outputs the subset cover $\mathcal{S}_c$.

1. Form the initial list $\mathcal{L}$ with all revoked leaf nodes at level 0 of $\mathcal{T}^0$. Mark each node $j \in \mathcal{L}$ as covered and set $\mathsf{SDnodes}[j] = \{j\}$.

2. Process nodes in $\mathcal{L}$ from left to right. If $\mathcal{L}[t]$ is the root node, go to step 3. Otherwise, let i be the parent of $\mathcal{L}[t]$. At the t^{th} iteration:

   (a) If $\mathcal{L}[t]$ and $\mathcal{L}[t+1]$ have the same parent, skip the step below and continue to the next iteration for $\mathcal{L}[t+1]$.

   (b) Else, append i to $\mathcal{L}$. Let $\{j_1, \ldots, j_c\}$ be the children of i in $\mathcal{L}$. The following mutually exclusive cases occur:

      i. Case when all nodes $j_1, \ldots, j_c$ are covered:
         A. If $c < k$, mark i as intermediate and set $\mathsf{SDnodes}[i] = \{j_1, \ldots, j_c\}$.
         B. For $c = k$, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.

      ii. Case when $c = 1$ and j_1 is intermediate:
      Mark i as intermediate and copy $\mathsf{SDnodes}[j_1]$ to $\mathsf{SDnodes}[i]$.

      iii. Case when $c > 1$ and there is at least one intermediate node in $\{j_1, \ldots, j_c\}$:
      For $j \in \{j_1, \ldots, j_c\}$ that is intermediate, add $S_{j,\mathsf{SDnodes}[j]}$ to the cover $\mathcal{S}_c$ and mark j as covered.

A. If $c < k$, mark i as intermediate and set $\mathsf{SDnodes}[i] = \{j_1, \ldots, j_c\}$.
B. For $c = k$, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.

3. If the root node is marked as intermediate, add $S_{0,\mathsf{SDnodes}[0]}$ to the cover $\mathcal{S}_c$.

When the iterations terminate at step 2 in the above algorithm, if the root is marked as covered, it implies that all privileged users have been covered and hence no more SD subsets are added in step 3. The subset cover $\mathcal{S}_c$ output by the algorithm is a collection of subsets of the form $S_{i,\mathsf{SDnodes}[i]}$.

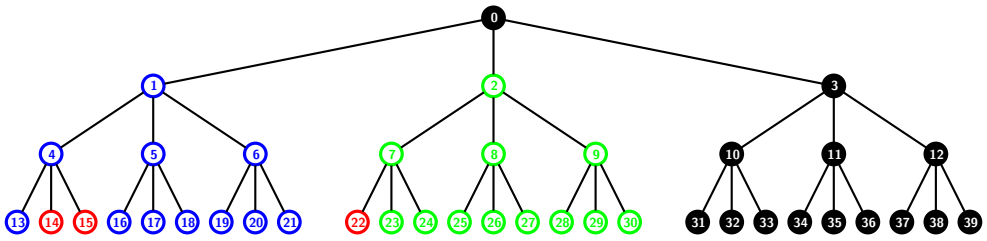

Figure 3: The subset cover $\mathcal{S}_c$ for $\mathcal{R} = \{14, 15, 22\}$ will contain the SD subsets $S_{1,\{14,15\}}$, $S_{2,\{22\}}$ and $S_{0,\{1,2\}}$.

Algorithm Demonstration To demonstrate the above algorithm, let us consider the revocation pattern $\mathcal{R} = \{14, 15, 22\}$ in the tree $\mathcal{T}^0$ with 27 users in Figure 3. The list $\mathcal{L}$ that is operated on iteratively in the algorithm, is eventually populated with the nodes $\{14, 15, 22, 4, 7, 1, 2, 0\}$. These are nodes that lie on the paths joining revoked users with the root node. Nodes 14, 15 and 22 are initially covered. The parent 4 of 14 and 15 is appended to the list and marked as intermediate with $\mathsf{SDnodes}[4] = \{14, 15\}$. Similarly, 7 is appended to the list and marked as intermediate with $\mathsf{SDnodes}[7] = \{22\}$. Next 1 is appended and marked as intermediate with $\mathsf{SDnodes}[4] = \{14, 15\}$ copied to $\mathsf{SDnodes}[1]$. Then 2 is appended and marked as intermediate with $\mathsf{SDnodes}[7] = \{22\}$ copied to $\mathsf{SDnodes}[2]$. Finally, 0 is appended to the list. Since 0 has two children in the list which are not covered, the subsets $S_{1,\mathsf{SDnodes}[1]}$ and $S_{2,\mathsf{SDnodes}[2]}$ are added to the cover $\mathcal{S}_c$. Node 0 is marked as intermediate with $\mathsf{SDnodes}[0] = \{1, 2\}$. Finally, the subset $S_{0,\mathsf{SDnodes}[0]}$ is added to $\mathcal{S}_c$. Hence, for $\mathcal{R} = \{14, 15, 22\}$, $\mathcal{S}_c = \{S_{1,\{14,15\}}, S_{2,\{22\}}, S_{0,\{1,2\}}\}$.

Once the subset cover $\mathcal{S}_c$ has been constructed, the message M is encrypted using a random session key, which in turn is encrypted for each set in the cover. These encryptions of the session key are sent along with the encrypted message as the header part of the cipher-text. The number of sets in the cover, also called the header length, is the parameter determining the transmission overhead of the scheme.

Nodes That Generate A Subset Nodes in $\mathcal{L}$ are the only nodes of $\mathcal{T}^0$ that are processed in the cover-finding algorithm. Hence, subsets in the cover are generated from nodes in $\mathcal{L}$ only. In other words, if $S_{j,\mathsf{SDnodes}[j]}$ is in the cover, then $j \in \mathcal{L}$ and $\mathsf{SDnodes}[j] \subset \mathcal{L}$. The following Lemma 1 identifies the properties of the node j and the set of nodes $\mathsf{SDnodes}[j]$.

Lemma 1. *Suppose* $S_{j,\mathsf{SDnodes}[j]}$ *is in the subset cover. Node* j *and the nodes in* $\mathsf{SDnodes}[j]$ *have the following properties:*
(1-a) Not all k *children of* j *are in* $\mathcal{L}$*, and*
(1-b) j *is either the root or an internal node with a sibling in* $\mathcal{L}$*.*
(2-a) If $\mathsf{SDnodes}[j] = \{v\}$*, then* v *is either a leaf node or an internal node with all its children in* $\mathcal{L}$*.*
(2-b) If $|\mathsf{SDnodes}[j]| > 1$*, then all nodes of* $\mathsf{SDnodes}[j]$ *are siblings.*
(2-c) For any node j*,* $|\mathsf{SDnodes}[j]| < k$*.*

Proof. First we show that j is either the root or an internal node with a sibling in $\mathcal{L}$. At step 3 of the cover finding algorithm described above, we see that SD subsets of the form $S_{0,\mathsf{SDnodes}[0]}$ may be generated. Hence, node j can be the root. If node j is not the root, then the only other way a subset may be generated is in step 2-b-iii of the algorithm. In this step, the algorithm considers a node i in $\mathcal{L}$ with a set $\{j_1, \ldots, j_c\}$ of its children in $\mathcal{L}$ where $c > 1$. Every $j \in \{j_1, \ldots, j_c\}$ that is marked as intermediate at that point, generates a subset. Hence, a non-root node j that generates a subset, must have a sibling in $\mathcal{L}$.

Next, we show that not all k children of j are in $\mathcal{L}$. The root node generates a subset in step 3 of the algorithm only if it is marked as intermediate. A node j that generates a subset in step 2-b-iii, is marked as intermediate until that point. Hence j is not marked covered until the subset $S_{j,\mathsf{SDnodes}[j]}$ is generated from it. This implies that in previous iterations, when the children of j in $\mathcal{L}$ were being processed, it was not marked as covered. A node may be marked as covered in either (1) step 2-b-i-B or 2-b-iii-B when all its children are in $\mathcal{L}$, or (2) step 2-b-iii after it has generated a subset. If $j = \mathcal{L}[t]$, then until the t^{th} iteration, j remains marked as intermediate if the number of children of j in $\mathcal{L}$ is less than k. Thus, not all k children of j are in $\mathcal{L}$.

In the cover finding algorithm, step 1 and the three mutually exclusive steps within step 2-b are the only places from where a set $\mathsf{SDnodes}[j]$ may arise. From step 1 of the algorithm, we see that for each leaf node j, $\mathsf{SDnodes}[j]$ is singleton. From steps 2-b-i-B and 2-b-iii-B, we see that if all k children of an internal node j are in $\mathcal{L}$, then $\mathsf{SDnodes}[j]$ is singleton. These are the only two ways in which $\mathsf{SDnodes}[j]$ for a node j can be singleton.

A set $\mathsf{SDnodes}[j]$ with more than one node is created only in steps 2-b-i-A and 2-b-iii-A. Clearly, the nodes in $\mathsf{SDnodes}[j]$ have a common parent i and hence are siblings of each other. Hence, if $|\mathsf{SDnodes}[j]| > 1$, then all nodes in $\mathsf{SDnodes}[j]$ are siblings. It is also clear from steps 2-b-i-A and 2-b-iii-A that $\mathsf{SDnodes}[j]$ can have at most k nodes. Hence, $|\mathsf{SDnodes}[j]| < k$ for any j. □

Correctness Of The Algorithm We prove that the algorithm described above, generates subsets that were assigned keys during initiation as has been described in Section 3.1. We also show that all privileged users are in some subset in $\mathcal{S}_c$ and no revoked user is included in any of the subsets.

Theorem 2. *A subset $S_{j,\mathsf{SDnodes}[j]}$ generated by the algorithm is such that (1) $1 \leq |\mathsf{SDnodes}[j]| < k$, (2) all nodes in $\mathsf{SDnodes}[j]$ are siblings of each other, and (3) j is their ancestor. The union of the subsets in $\mathcal{S}_c$ include all privileged leaves and no revoked leaves.*

Proof. From Lemma 1 we know that all nodes in $\mathsf{SDnodes}[j]$ are siblings of each other and $1 \leq |\mathsf{SDnodes}[j]| < k$. It can be seen from steps 1 and 2-b of the algorithm that a node j_1 gets inserted into a set $\mathsf{SDnodes}[j]$ only if j is an ancestor of j_1.

A subset $S_{i,\mathsf{SDnodes}[i]}$ output by the algorithm, represents all leaf nodes in the induced subgraph

$$\mathcal{T}^i \setminus \cup_{j \in \mathsf{SDnodes}[i]} \mathcal{T}^j.$$

In other words, the subset $S_{i,\mathsf{SDnodes}[i]}$ has leaves in $\mathcal{T}^i$ that are not in the subtrees in $\mathsf{SDnodes}[i]$. It can be seen from steps 2-b-i and 2-b-iii that a covered node in $\mathcal{L}$ is always within some subtree in the set SDnodes of its parent and ancestors thereon. Hence, once marked covered, a node is not in any set $S_{i,\mathsf{SDnodes}[i]}$ in $\mathcal{S}_c$ that is included thereafter. From steps 1 and 2-b-1 of the cover finding algorithm, we know that each revoked leaf j is in $\mathsf{SDnodes}[j]$ and hence in some subtree in $\mathsf{SDnodes}[i]$ for every ancestor i of j. This implies that a revoked leaf cannot be in any subset in $\mathcal{S}_c$.

We next show that any privileged leaf is in a subset in $\mathcal{S}_c$. Let us consider the path joining a privileged leaf to the root in $\mathcal{T}^0$. Since the root node is in $\mathcal{L}$, hence there will be at least one node on this path that is in $\mathcal{L}$. Let j_1 be the node on this path that is in $\mathcal{L}$ and is nearest to the privileged leaf. All subsequent nodes above j_1 are in $\mathcal{L}$. Again, since the root node is on the path $(j_1, \ldots, 0)$, at least one of the nodes on this path generate a subset in $\mathcal{S}_c$. Let the node in $(j_1, \ldots, 0)$ that is nearest to the privileged leaf and generates a subset be j. Either $j = j_1$ or j is an ancestor of j_1 on the path $(j_1, \ldots, 0)$. The subset $S_{j,\mathsf{SDnodes}[j]}$ generated from node j has

all leaves in $\mathcal{T}^j$ but not in any of the subtrees in $\mathsf{SDnodes}[j]$. The privileged leaf is in $\mathcal{T}^j$. We show that it is not in any of the subtrees in $\mathsf{SDnodes}[j]$. From steps 1 and 2-b in the algorithm, we know that $\mathsf{SDnodes}[j]$ has nodes that have been covered. Nodes between the privileged leaf and before j_1 are not in $\mathcal{L}$ and hence cannot be covered. Since j is the only node on the path $(j_1, \ldots, j)$ that generates a subset, all nodes on this path are intermediate until the subset $S_{j,\mathsf{SDnodes}[j]}$ is generated from j. Consequently, $\mathsf{SDnodes}[j]$ does not have any of the nodes on the path joining the privileged leaf and j. Hence, the privileged leaf is not in any of the subtrees rooted at nodes in $\cup_{j' \in \mathsf{SDnodes}[j]} \mathcal{T}^{j'}$. Hence, the privileged user is in the subset $S_{j,\mathsf{SDnodes}[j]}$. □

Relation To The NNL-SD Scheme The case $k = 2$ of the above scheme is exactly the NNL-SD scheme described in [NNL01, NNL02]. So, the new scheme is a generalization of the NNL-SD scheme and subsumes it. One important advantage of the new scheme is a uniform description of the cover generation algorithm irrespective of the value of k. We note that this description is simpler than the description for $k = 2$ given in [NNL01, NNL02] which is phrased in terms of Steiner trees. It turns out that the more elementary description of the cover generation algorithm is cleaner which leads to an easier implementation.

Relation To The Ternary Tree Scheme In [FKTS08] For $k = 3$, the collection $\mathcal{S}$ that we consider is the same as that in [FKTS08]. However, the method for assigning keys to these subsets is different. The work [FKTS08] uses a hash chain method and mentions that it does not extend to k-ary trees for $k \geq 4$. On the other hand, our method of distributing secret keys to the users is general and works for all k. In Section 6, we describe a modified method of distributing secret keys which further lowers the user storage requirement.

3.3 Traitor Tracing

Traitor tracing is the mechanism to identify leaked user keys from a pirate decoder by treating it as a "black-box". Traitor tracing for the NNL-SD scheme was discussed in details in [NNL01, NNL02]. They showed that traitor tracing can be done on any scheme that assigns keys to subsets which satisfy the *bifurcation property*. The bifurcation property states that *given any subset that is in the collection $\mathcal{S}$ and hence has been assigned a key, it is possible to partition the set into two (or a constant number of) almost equal subsets from $\mathcal{S}$.* The *bifurcation value* is defined to be the ratio of the size of the largest subset to that of the set itself.

For the k-ary tree SD scheme, a subset $S_{i,J}$ in its collection $\mathcal{S}$ is such that all nodes in the set J are siblings and are in the subtree $\mathcal{T}^i$. If the parent of the nodes in J is i, then the subset $S_{i,J}$ is split into equal sized subsets $\mathcal{T}^j$ where j is a child of i and $j \notin J$. Thus, each child subtree of i whose root is not in J forms a subset in the split. All the subsets formed by splitting $S_{i,J}$ will be of equal size and hence the bifurcation value in this case is $1/(k - |J|)$. The worst case (maximum bifurcation value) occurs when there are two child subtrees of i that are not in J (and are hence privileged). The maximum bifurcation value is $1/2$. If the parent of nodes in J is a descendant of i, then the subset $S_{i,J}$ will be split into exactly k subsets each formed from a child subtree of node i. There will be one subset formed from the child subtree of i that contains the nodes in J. This subset will be smaller than the rest of the $k - 1$ equal-sized subsets. The bifurcation value in this case will be $1/k$. All subsets in the collection $\mathcal{S}$ of the layered k-ary tree SD scheme belong to the collection of subsets that are assigned keys in the k-ary tree SD scheme. Hence, the bifurcation property also holds for those subsets. Thus, a traitor tracing mechanism can be devised for the scheme introduced in this work in a manner similar to the one described in [NNL01, NNL02].

The number of queries required by the traitor tracing algorithm depends on the bifurcation value. At every step of the traitor tracing algorithm, a subset S of users that contains a traitor is divided into subsets $S_1, \ldots, S_t$ using the bifurcation property as mentioned above. Each subset S_t is tested for containment of a traitor. The ratio $|S_t|/|S|$ is at most the bifurcation value. Lesser the bifurcation value, lesser is the size of the remaining subset from which the traitors have to be traced. Hence, the traitor tracing algorithm will be more efficient. The bifurcation value of the NNL-SD scheme is $2/3$. The bifurcation value of the k-ary tree SD scheme is $1/2$

for $k \geq 3$. Hence, the traitor tracing mechanism for the k-ary tree SD scheme will be more efficient than the NNL-SD scheme.

4 Header Length Analysis

When there are no revoked users, the header length is 1. Henceforth, we assume the set $\mathcal{R}$ of revoked users to be non-empty.

Theorem 3. *Fix $k \geq 2$, $n \geq 1$ and $1 \leq r \leq n$. Then the maximum header length that can be achieved is* $\min(2r-1, n-r, n/k)$.

Note: When r is small, the bound $2r-1$ applies. For $k=2$, the upper bound of $2r-1$ was given in [NNL01, NNL02]. The more general form of the bound for binary trees was mentioned in [BS13]. For $k=3$, it has been shown that the scheme in [FKTS08] has an upper bound of $\min(2r-1, n/3)$.

Proof. The bound $n-r$ on the header length arises since each of the $n-r$ privileged users can be covered by singleton subsets in the header.

The bound n/k is obtained as follows. Suppose the header consists of h subsets. Write $h = h_1 + \cdots + h_{k-1} + h_k$ where for $1 \leq i \leq k-1$, h_i is the number of subsets in the header having exactly i privileged users and h_k is the number of subsets in the header having at least k privileged users.

Suppose S is a subset counted in h_i for some i in $[1, k-1]$. From the cover finding algorithm, it necessarily follows that the leaf nodes in S are siblings and the other siblings of the nodes in S are revoked. So, for each subset S counted in h_i, there corresponds a total of k users (i users in S and the other $k-|S|$ revoked siblings of the users in S). As a result, the total number of users accounted for by $h_1, \ldots, h_{k-1}$ is $k(h_1 + \cdots + h_{k-1})$. Since each subset counted in h_k has at least k users, the total number of users n is at least $k(h_1 + \cdots + h_{k-1}) + kh_k = k(h_1 + \cdots + h_{k-1} + h_k) = kh$. From this it follows that $h \leq n/k$.

Now, we turn to the bound $2r-1$. The subtree $\mathcal{T}^j$ may be written as a union of all its child subtrees. Hence,

$$\mathcal{T}^j = \bigcup_{j' \in \{kj+1, \ldots, kj+k\}} \mathcal{T}^{j'}.$$

Thus, the node j can be replaced by $\{kj+1, \ldots, kj+k\}$. For a subset $S_{i,J} \in \mathcal{S}_c$, if $j \in J$ such that all k children of j are in $\mathcal{L}$, we replace j with $\{kj+1, \ldots, kj+k\}$ in J to get J'.

$$J' = (J \setminus \{j\}) \cup \{kj+1, \ldots, kj+k\}.$$

We keep replacing nodes in J' having k children in $\mathcal{L}$ by their children until all nodes in J' have less than k children in $\mathcal{L}$. Some nodes in J' may have no children (leaf nodes) in $\mathcal{L}$. These are revoked leaf nodes of $\mathcal{T}^0$ that were inserted in $\mathcal{L}$ in step 1 of the algorithm. The new representation $\mathcal{S}'_c$ of the subset cover $\mathcal{S}_c$ will have

$$\mathcal{S}'_c = (\mathcal{S}_c \setminus S_{i,J}) \cup S_{i,J'}.$$

However, the privileged users in $S_{i,J'}$ are exactly the privileged users in $S_{i,J}$. We do this for all subsets in $\mathcal{S}_c$ to complete the new representation $\mathcal{S}'_c$ of $\mathcal{S}_c$.

We first show that all internal nodes in J' generate a subset each. From Lemma 1 we know that for $S_{i,J} \in \mathcal{S}_c$, all nodes in J are in $\mathcal{L}$ and are siblings. During the transformation, a node $j \in J$ is replaced by k nodes which are also in $\mathcal{L}$ and are siblings of each other. Hence, each node $j \in J'$ has a sibling in $\mathcal{L}$. Since j is in $\mathcal{L}$ and has less than k children, hence from step 2-b of the algorithm we know that it is marked as intermediate unless a subset is generated from it. From step 2-b-iii we know that an intermediate node having a sibling in $\mathcal{L}$, generates a subset. Hence, a node in J' is either a revoked leaf node or an internal node that generates a subset.

We construct a graph Υ such that for each subset $S_{i,J'}$ in $\mathcal{S}'_c$, node i and all nodes in J' are in Υ. For every subset $S_{i,J'}$ in $\mathcal{S}'_c$, there is an edge (i, j) for each $j \in J'$ in Υ. A node $j \in J'$ that is an internal node in $\mathcal{T}^0$ generates a subset and hence is an internal node in Υ. A leaf node in J' is a leaf node in Υ.

We first show that Υ is a forest with one or more component trees. Once a subset $S_{i,J}$ is included in $\mathcal{S}_c$ at step 2-b-iii of the algorithm, i is marked as covered and $\mathsf{SDnodes}[i] = i$. Hence for an ancestor i_1 of i, any descendant of i is not in $\mathsf{SDnodes}[i_1]$. If S_{i_1,J_1} is included in the cover, it may have i in J_1. Since i generates a subset in the cover, the transformation of J_1 to J'_1 will not reach any descendant of i. Hence, J'_1 will not have any descendant of i. Consequently, there will be an edge (i, j) in Υ for each $j \in J'$ but there will be no other edge between j and any other ancestor of j. Since this is true for any node $j \in \Upsilon$, it is an acyclic graph. Additionally, the cover might not have a subset generated from the root. Thus, components of Υ may not be connected. Hence, it is a forest with one or more component trees.

The nodes in Υ are either internal nodes in $\mathcal{T}^0$ that generate a subset each, or revoked leaf nodes. Hence, the number of internal nodes in Υ is the number of subsets in the subset cover. For a subset $S_{i,J} \in \mathcal{S}_c$ if $|J| = 1$, then by Lemma 1 the node in J is either an internal node with all its k children in $\mathcal{L}$ or a leaf node. In the corresponding $S_{i,J'} \in \mathcal{S}'_c$, an internal node in $j \in J$ with all its k children in $\mathcal{L}$ has been replaced by its child nodes. Hence, if a subset $S_{i,J'} \in \mathcal{S}'_c$ is such that $|J'| = 1$, then the set J' has a single leaf node. Hence, if an internal node i in Υ has only one child, it will be a leaf. Any other internal node in Υ will have at least two children in Υ.

The transformation ensures that each of the r revoked leaves of $\mathcal{T}^0$ is a leaf in Υ. Hence, there can be at most r internal nodes that have leaf nodes amongst their children in Υ. The graph Υ is reduced to Υ' by merging an internal node having a single leaf child, with its child. Consequently, Υ' is a forest with at most r leaves and internal nodes in Υ' have at least two children each. Hence, there are at most $r - 1$ internal nodes in Υ'. Thus, the maximum number of internal nodes in Υ is $r + r - 1 = 2r - 1$. Hence, there can be at most $2r - 1$ subsets in the subset cover. □

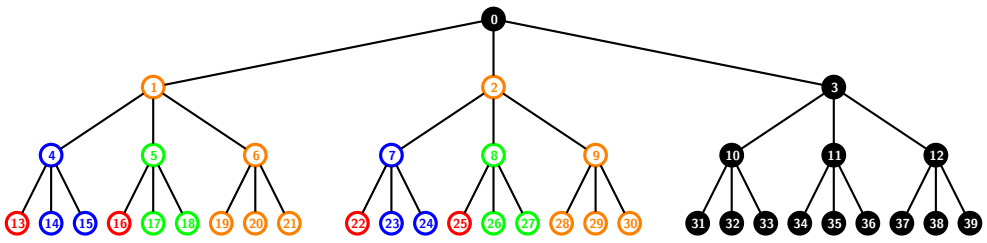

Figure 4: Example showing that the upper bound of $2r - 1$ on the header length is tight for $k = 3$. The subset cover for $\mathcal{R} = \{13, 16, 22, 25\}$ in the tree $\mathcal{T}^0$ with $k = 3$, will contain the SD subsets $S_{4,\{13\}}$, $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$.

This upper bound of $2r - 1$ on the maximum header length can be achieved for a given r and any fixed value of k provided n can be made as large as required. For $k = 2$, this bound has been shown to be tight in [BS13]. For $k = 3$, let us consider the tree $\mathcal{T}^0$ of Figure 4 where the set of revoked users is $\mathcal{R} = \{13, 16, 22, 25\}$. The nodes in $\mathcal{P}$ are $\{13, 16, 22, 25, 1, 2, 0\}$. The subsets in the cover are $S_{4,\{13\}}$, $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$. It can be seen from this figure that for higher arities (> 3), additional subtrees are added to all the internal nodes. Assuming that the revoked users remain the same as marked in the figure, we notice the following. The subset $S_{4,\{13\}}$ gets additional users that are attached to the node 4. Similarly, each of the subsets $S_{5,\{16\}}$, $S_{7,\{22\}}$, $S_{8,\{25\}}$, $S_{1,\{4,5\}}$, $S_{2,\{7,8\}}$ and $S_{0,\{1,2\}}$ get the additional users attached to the nodes $5, 7, 8, 1, 2$ and 0 respectively. Hence, this upper bound is tight for any arity k in general provided n can be chosen to be large.

For a given k and $n = k^{\ell_0}$ this maximum header length of $2r - 1$ is achieved for r given by Lemma 4.

Lemma 4. *For a given k and $n = k^{\ell_0}$, the maximum header length of $2r-1$ is achieved for $r = 2^{\ell_0 - 1}$.*

Proof. We prove this by induction on ℓ_0. For $\ell_0 = 1$, $n = k$ and $r = 1$. Let j_1 be the only revoked leaf. The only subset in the cover is $S_{0,\{j_1\}}$ and hence the header length is $2r - 1 = 1$. We assume that the maximum header length that can be achieved for $2^{\ell_0 - 2}$ revoked users in a full tree of arity k and with $k^{\ell_0 - 1}$ users is $2^{\ell_0 - 1} - 1$. Let us consider a tree with $n = k^{\ell_0}$ users and $r = 2^{\ell_0 - 1}$ revoked users such that two of the subtrees (out of k) of the root node, which are rooted at nodes j_1 and j_2, have $r/2 = 2^{\ell_0 - 2}$ revoked users in each and the rest of the subtrees of the root node do not have any revoked user in them. Since each of these two subtrees have $k^{\ell_0 - 1}$ users in each, hence by assumption they give rise to $2^{\ell_0 - 1} - 1$ subsets each in the cover. Additionally, there will be a subset $S_{0,\{j_1,j_2\}}$ in the cover. Hence, the total number of subsets generated by this construction for $r = 2^{\ell_0 - 1}$ is $2 \times (2^{\ell_0 - 1} - 1) + 1 = 2r - 1$.

We need to show that these subsets do not combine to reduce the header length. Two SD subsets $S_{i_1,\mathsf{SDnodes}[i_1]}$ and $S_{i_2,\mathsf{SDnodes}[i_2]}$ can be combined into one SD subset if (1) $\mathsf{SDnodes}[i_1] = \{i_2\}$ or $\mathsf{SDnodes}[i_2] = \{i_1\}$ or (2) if nodes in $\mathsf{SDnodes}[i_1]$ are siblings of nodes in $\mathsf{SDnodes}[i_2]$. Any two SD subsets $S_{i_1,\mathsf{SDnodes}[i_1]}$ and $S_{i_2,\mathsf{SDnodes}[i_2]}$ from the subtrees rooted at j_1 and j_2 respectively cannot satisfy either of the above two conditions. For the same reason, the subset $S_{0,\{j_1,j_2\}}$ cannot be combined with any of the SD subsets in these two subtrees. Hence, none of these subsets combine to reduce the total number of subsets $2r - 1$ in the subset cover. Hence, for $r = 2^{\ell_0 - 1}$, the maximum header length of $2r - 1$ is achieved for a fixed k and $n = k^{\ell_0}$. □

Effect Of k On The Header Length In the Subset-Cover framework, the subsets in the collection $\mathcal{S}$ are used to cover the privileged users. It seems intuitive that if the number of subsets in the collection increases, the number of subsets required to form a cover would decrease. As mentioned earlier, for $n = 16$, the number of subsets in the collection for $k = 4$ is more than that for $k = 2$. More subsets in the collection should reduce the header length. In Figure 5 we see that the header length due to the given revocation pattern is smaller for the 4-ary tree as compared to the 2-ary one. However, this may not always happen. In Figure 6, the revocation pattern is such that the header length is smaller for the 2-ary tree as compared to the 4-ary one. Hence, we see that the header length may not always reduce by increasing the arity. By Theorem 3, the maximum header length is independent of the underlying arity k. The expected header lengths for different values of k will give a better idea about the effect of arity on the overall communication overhead.

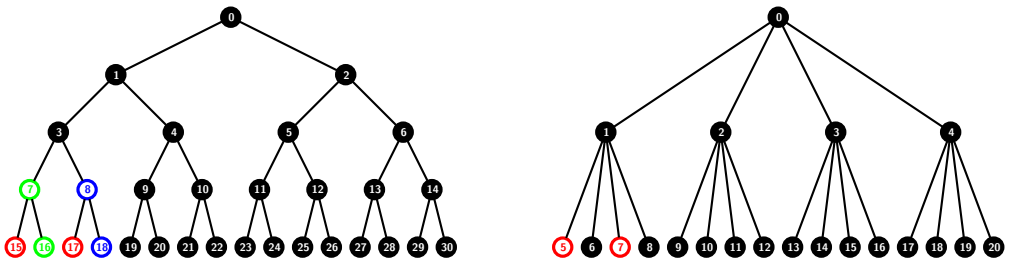

Figure 5: Example where the header length of 4-ary is better than 2-ary: For $n = 16$ users, the header length for $\mathcal{R} = \{15, 17\}$ in the 2-ary tree is more for $k = 2$ than for $k = 4$. For $k = 2$, the subset cover $S_c = \{S_{7,\{15\}}, S_{8,\{17\}}, S_{0,\{3\}}\}$. For $k = 4$, the subset cover $S_c = \{S_{0,\{5,7\}}\}$.

4.1 Expected Header Length

Fix k, n and r. Consider the following random experiment. Randomly choose r out of the n users uniformly at random one-by-one and without replacement. Consider the selected set of r users to be revoked. The expected header length under this random experiment is given by the following result.

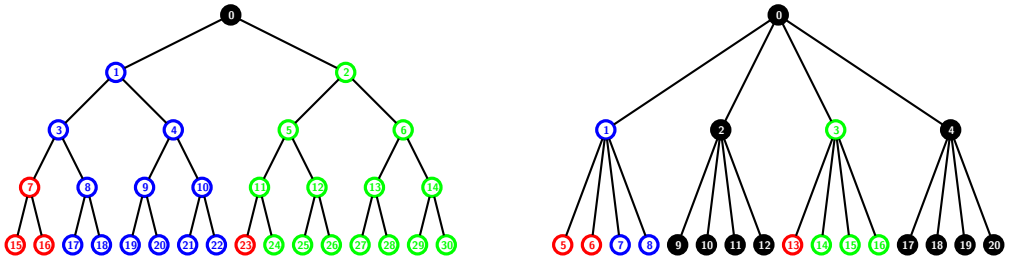

Figure 6: Example where the header length of 2-ary is better than 4-ary: For $n = 16$ users, the header length for $\mathcal{R} = \{15, 16, 23\}$ in the 2-ary tree is more for $k = 4$ than for $k = 2$. For $k = 2$, the subset cover $S_c = \{S_{1,\{7\}}, S_{2,\{23\}}\}$. For $k = 4$, the subset cover $S_c = \{S_{1,\{5,6\}}, S_{3,\{13\}}, S_{0,\{1,3\}}\}$.

Theorem 5. *Fix $k \geq 2$, $n = k^{\ell_0} \geq 1$ and $1 \leq r \leq n$. The expected header length in the k-ary tree SD scheme is given by*

$$\sum_{c=1}^{k-1} \binom{k}{c} \left(\gamma_{\ell_0,c} + \sum_{\ell=1}^{\ell_0-1} (k^{\ell_0-\ell}) \gamma_{\ell,c} \right)$$

where

$$\begin{aligned} \gamma_{\ell,c} &= \eta_r(n, k^\ell - ck^{\ell-1}) - \eta_r(n, k^{\ell+1} - ck^{\ell-1}) \\ &\quad - \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^\ell - (c-t)k^{\ell-1}) \\ &\quad + \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell+1} - (c-t)k^{\ell-1}) \qquad \text{for } 1 \leq \ell \leq \ell_0 - 1 \end{aligned} \tag{3}$$

and

$$\gamma_{\ell_0,c} = \eta_r(n, k^{\ell_0} - ck^{\ell_0-1}) - \sum_{t=1}^{c} (-1)^{t+1} \binom{c}{t} \eta_r(n, k^{\ell_0} - (c-t)k^{\ell_0-1}). \tag{4}$$

Proof. Let $X_{n,r}$ be the random variable taking the value of the header length. The expected header length also depends on k and so strictly speaking we should be using the notation $X_{n,r,k}$ to denote this dependence. We have chosen the simpler notation $X_{n,r}$ since for a particular implementation, k will be fixed and so clear from the context.

Each subset in the cover $\mathcal{S}_c$ is rooted at some internal node i of $\mathcal{T}^0$. Each such node in the tree contributes at most one subset to the cover. Let $X^i_{n,r}$ be the random variable associated with node i, that denotes its contribution to the header length. Hence, $X^i_{n,r} \in \{0, 1\}$. The event $X^i_{n,r} = 1$ occurs when there is a subset $S_{i,J}$ in the cover and $X^i_{n,r} = 0$ otherwise. It can be seen from the cover finding algorithm described in Section 3.2 that the leaf nodes of $\mathcal{T}^0$ do not generate SD subsets. Hence, SD subsets are generated only from the internal nodes in $\mathcal{T}^0$. It follows that

$$X_{n,r} = X^0_{n,r} + X^1_{n,r} + \cdots + X^{i_f}_{n,r} \tag{5}$$

where $i_f = \frac{nk-1}{k-1} - n - 1$ is the last internal node as per the labeling of the tree $\mathcal{T}^0$. By linearity of expectation,

$$E[X_{n,r}] = E[X^0_{n,r}] + E[X^1_{n,r}] + \cdots + E[X^{i_f}_{n,r}]. \tag{6}$$

To find the expected header length, one needs to compute the values of $E[X^i_{n,r}]$ for each $i \in \{0, 1, \ldots, i_f\}$. Since $X^i_{n,r} \in \{0, 1\}$, the random variable $X^i_{n,r}$ follows *Bernoulli distribution*. Hence, $E[X^i_{n,r} = 1] = \Pr[X^i_{n,r} = 1]$. Thus, to compute the expected header length $E[X_{n,r}]$ for a random revocation pattern, the probability $\Pr[X^i_{n,r} = 1]$ that a node i generates a subset has to be computed.

Let I be the set of all child nodes of i and let p be the parent node of i in $\mathcal{T}^0$. Hence, $I = \{ki+1, \ldots, ki+k\}$ and $p = \lfloor (i-1)/k \rfloor$. Let $J \subset I$ be a non-empty subset of child nodes of i. The event $X^i_{n,r} = 1$ occurs when a subset $S_{i,J}$ is in the cover. For a subset $S_{i,J}$ to be in the cover, the following have to be true with respect to node i in $\mathcal{T}^0$:

- At least one but not all child subtrees of i would contain some revoked nodes; and
- If $i \neq 0$ (for a non-root internal node), at least one sibling subtree of i would contain a revoked node.

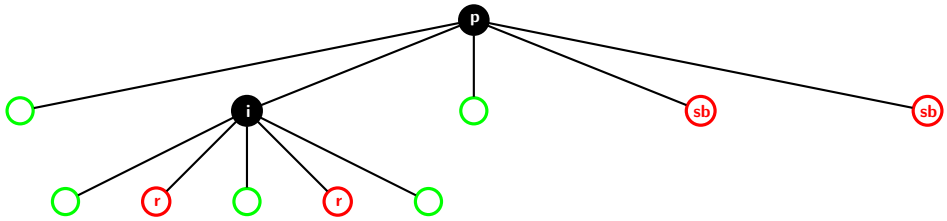

Figure 7: Example of a scenario for the event $X^i_{n,r} = 1$ where a set $S_{i,J}$ occurs in the subset cover. Child nodes of i marked with r are roots of subtrees containing at least one revoked user in each. Hence they are in the set J. The other child subtrees of i do not contain any revoked user. Hence they are in the set $I \setminus J$. Sibling subtrees of i marked as sb contain at least one revoked user in each. Hence, at least one child subtree of i (not all) has revoked users and at least one sibling subtree of i has revoked users. Consequently, a subset rooted at node i is generated.

In order to formulate these conditions when a subset $S_{i,J}$ rooted at node i occurs in the subset cover, we define some additional events with respect to the node i and a non-empty subset of its child nodes $J \subset I$. The event R^i_j (where $j \in J$) is defined to occur when for a revocation pattern, the subtree rooted at node j contains at least one revoked user. Hence, event $\overline{R^i_j}$ occurs when the subtree rooted at node j does not contain any revoked user. Let $R^i_J = \bigwedge_{j \in J} R^i_j$ be the event where each subtree rooted at nodes in J has at least one revoked user. Hence, $\overline{R^i_J}$ is the event where none of the subtrees rooted at nodes in J have any revoked user. For $i \neq 0$, event R^i_{sb} is defined to occur when the union of all sibling subtrees of i (children of p other than i) contains at least one revoked node. Hence, a subset $S_{i,J}$ is in the subset cover when the following condition is true

$$\left(\bigwedge_{j \in J} R^i_j \right) \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}} = R^i_J \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}}.$$

Since a subset $S_{i,J}$ may occur for any non-empty set $J \subset I$, hence the event $X^i_{n,r} = 1$ can also be written as

$$\bigvee_{J \subset I, J \neq \phi} \left(R^i_J \wedge R^i_{sb} \wedge \overline{R^i_{I \setminus J}} \right).$$

These events $(R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}})$ are mutually exclusive and they exhaustively form the event $X^i_{n,r} = 1$. Hence, we can write

$$\Pr[X^i_{n,r} = 1] = \sum_{J \subset I; J \neq \phi} \Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I \setminus J}}]. \tag{7}$$

It can be similarly seen that for the root node with children in $\{1, \ldots, k\}$

$$\Pr[X^0_{n,r}] = \sum_{J \subset \{1,\ldots,k\}; J \neq \phi} \Pr[R^i_J \wedge \overline{R^i_{I\setminus J}}]. \tag{8}$$

The probability $\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}]$ can be written as

$$\begin{aligned}
&\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] \\
&= \Pr[R^i_{sb} \wedge R^i_J | \overline{R^i_{I\setminus J}}] \times \Pr[\overline{R^i_{I\setminus J}}] \\
&= (1 - \Pr[\overline{R^i_{sb} \wedge R^i_J} | \overline{R^i_{I\setminus J}}]) \times \Pr[\overline{R^i_{I\setminus J}}] \\
&= (1 - \Pr[\overline{R^i_{sb}} | \overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_J} | \overline{R^i_{I\setminus J}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} | \overline{R^i_{I\setminus J}}]) \times \Pr[\overline{R^i_{I\setminus J}}] \\
&= \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}])
\end{aligned} \tag{9}$$

and for the root node, $\Pr[R^i_J \wedge \overline{R^i_{I\setminus J}}]$ can be written as

$$\Pr[R^i_J \wedge \overline{R^i_{I\setminus J}}] \quad = \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}]. \tag{10}$$

For the computation of (9) and (10) above, we observe that the event $\overline{R^i_J}$ occurs when at least one $j \in J$ does not contain any revoked user. Now let us consider two events A and B in general such that the event A occurs when all the sub-events $A_1, \ldots, A_c$ occur. Hence, $A = A_1 \wedge \ldots \wedge A_c$.

$$\begin{aligned}
\Pr[\overline{A} \wedge \overline{B}] \quad &= \Pr[\overline{A_1 \wedge \ldots \wedge A_c} | \overline{B}] \Pr[\overline{B}] \\
&= \Pr[\overline{A_1} \vee \ldots \vee \overline{A_c} | \overline{B}] \Pr[\overline{B}] \\
&= \Pr[\overline{A_1} \wedge \overline{B}] + \ldots + \Pr[\overline{A_c} \wedge \overline{B}] \\
&\quad - \Pr[\overline{A_1} \wedge \overline{A_2} \wedge \overline{B}] - \ldots - \Pr[\overline{A_{c-1}} \wedge \overline{A_c} \wedge \overline{B}] + \ldots \\
&\quad + (-1)^{c+1} \left(\Pr[\overline{A_1} \wedge \ldots \wedge \overline{A_c} \wedge \overline{B}] \right).
\end{aligned} \tag{11}$$

Now, when $A = R^i_J$ where $A_t = R^i_{j_t}$ and $B = R^i_{I\setminus J}$, we get

$$\begin{aligned}
\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] \quad &= \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}}] + \ldots + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}}] - \ldots - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \ldots + (-1)^{c+1} \left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \right).
\end{aligned} \tag{12}$$

Similarly, for the expression $\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}]$ in (9) we get the following using the result in (11)

$$\begin{aligned}
\Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \quad &= \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] - \ldots \\
&\quad - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots \\
&\quad + (-1)^{c+1} \left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \right).
\end{aligned} \tag{13}$$

From (9), (12) and (13) we get

$$\begin{aligned}
\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] &= \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_J} \wedge \overline{R^i_{I\setminus J}}]) \\
&= \Pr[\overline{R^i_{I\setminus J}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}}] - \ldots - \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad + \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}}] + \ldots \\
&\quad + \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}] \\
&\quad - \ldots + (-1)^{c+2} \left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}}]\right) \\
&\quad + \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots \\
&\quad + \Pr[\overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad - \Pr[\overline{R^i_{j_1}} \wedge \overline{R^i_{j_2}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] \\
&\quad - \Pr[\overline{R^i_{j_{c-1}}} \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}] + \ldots \\
&\quad + (-1)^{c+1} \left(\Pr[\overline{R^i_{j_1}} \wedge \ldots \wedge \overline{R^i_{j_c}} \wedge \overline{R^i_{I\setminus J}} \wedge \overline{R^i_{sb}}]\right).
\end{aligned} \tag{14}$$

To find these probabilities, we define $\eta_r(n, x)$ to be the probability that r out of n elements are chosen uniformly at random without replacement but x out of these n elements never get chosen. In other words,

$$\eta_r(n, x) = \frac{\binom{n-x}{r}}{\binom{n}{r}}. \tag{15}$$

Let the number of users in the subtree rooted at node i be λ^i. Hence, the number of users in all the sibling subtrees of i is $\lambda^p - \lambda^i$. Hence, the sum of the number of users in the subtrees rooted at nodes in $I \setminus J$ is $\lambda^i - \sum_{j\in J} \lambda^j$. From (14) and (15) we get

$$\begin{aligned}
&\Pr[R^i_{sb} \wedge R^i_J \wedge \overline{R^i_{I\setminus J}}] \\
&= \eta_r(n, \lambda^i - \sum_{j\in J} \lambda^j) - \eta_r(n, \lambda^p - \sum_{j\in J} \lambda^j) \\
&\quad - \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1\}} \lambda^j) - \ldots - \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_c\}} \lambda^j) \\
&\quad + \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1,j_2\}} \lambda^j) + \ldots + \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_{c-1},j_c\}} \lambda^j) \\
&\quad - \ldots + (-1)^{c+2} \eta_r(n, \lambda_i) \\
&\quad + \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_1\}} \lambda^j) + \ldots + \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_c\}} \lambda^j) \\
&\quad - \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_1,j_2\}} \lambda^j) - \ldots - \eta_r(n, \lambda^p - \sum_{j\in J\setminus\{j_{c-1},j_c\}} \lambda^j) \\
&\quad + \ldots + (-1)^{c+1} \eta_r(n, \lambda_p).
\end{aligned} \tag{16}$$

Similarly for the root node, from (10), (12) and (15) we get

$$\begin{aligned}
&\Pr[R_J^i \wedge \overline{R_{I\setminus J}^i}] \\
&= \eta_r(n, \lambda^i - \sum_{j\in J} \lambda^j) \\
&\quad -\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1\}} \lambda^j) - \ldots - \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_c\}} \lambda^j) \\
&\quad +\eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_1,j_2\}} \lambda^j) + \ldots + \eta_r(n, \lambda^i - \sum_{j\in J\setminus\{j_{c-1},j_c\}} \lambda^j) \\
&\quad - \ldots + (-1)^{c+2}\eta_r(n, \lambda_i). \qquad (17)
\end{aligned}$$

From (6), (7), (8), (16) and (17) we get the algorithm for computing $E[X_{n,r}]$. In Section 5, we discuss that this scheme may be further extended for arbitrary number of users (instead of a power of k). In such a case, the underlying tree may be assumed to be a complete tree (with users at the last two levels of the tree) instead of a full tree. All the above expressions for computing probabilities are also valid for complete trees where the number of users may not be a power of k.

However, for the k-ary tree SD scheme as it has been described here, the number of users is assumed to be a power of k. Hence, $n = k^{\ell_0}$. Let $c = |J|$ be the cardinality of the set J of some child nodes of i. Hence, $0 < c < k$. There are ℓ_0 levels with internal nodes in the tree $\mathcal{T}^0$. All subtrees rooted at level ℓ has the same number of leaf nodes k^ℓ. For a non-root i at level ℓ and a corresponding set of child nodes J ($|J| = c$), the contribution to the header can be computed from (16) as

$$\begin{aligned}
\Pr[R_{sb}^i \wedge R_J^i \wedge \overline{R_{I\setminus J}^i}] &= \eta_r(n, k^\ell - ck^{\ell-1}) - \eta_r(n, k^{\ell+1} - ck^{\ell-1}) \\
&\quad -\sum_{t=1}^{c}(-1)^{t+1}\binom{c}{t}\eta_r(n, k^\ell - (c-t)k^{\ell-1}) \\
&\quad +\sum_{t=1}^{c}(-1)^{t+1}\binom{c}{t}\eta_r(n, k^{\ell+1} - (c-t)k^{\ell-1}). \qquad (18)
\end{aligned}$$

Let $\gamma_{\ell,c}$ be the value of this probability given by (18) for a non-root node i at level ℓ ($1 \le \ell \le \ell_0 - 1$) and $|J| = c$. Since there are $k^{\ell_0-\ell}$ nodes at level ℓ in the tree $\mathcal{T}^0$, hence the contribution of all the nodes at level ℓ and a fixed value of c is $(k^{\ell_0-\ell})\gamma_{\ell,c}$. For the root node 0 at level ℓ_0 and a corresponding set of child nodes $J \subset \{1, \ldots, k\}$ ($|J| = c$), the contribution to the header can be computed from (17) as

$$\begin{aligned}
&\Pr[R_J^0 \wedge \overline{R_{I\setminus J}^0}] \\
&= \eta_r(n, k^{\ell_0} - ck^{\ell_0-1}) - \sum_{t=1}^{c}(-1)^{t+1}\binom{c}{t}\eta_r(n, k^{\ell_0} - (c-t)k^{\ell_0-1}). \qquad (19)
\end{aligned}$$

The value of this probability given by (19) for the root node at level ℓ_0 and $|J| = c$ is denoted by $\gamma_{\ell_0,c}$. Hence, the expected header length for a given k, ℓ_0 and r is given by

$$E[X_{n,r}] = \sum_{c=1}^{k-1}\binom{k}{c}\left(\gamma_{\ell_0,c} + \sum_{\ell=1}^{\ell_0-1}(k^{\ell_0-\ell})\gamma_{\ell,c}\right). \qquad (20)$$

□

Table 1: Table showing the performance of the algorithm for computing the expected header length. For each k, we have chosen n to be k^a and k^b which are the two closest powers of k to 10^8. The column MHL_k/r gives the ratio of the mean header length MHL_k for k-ary tree to the number of revoked users r.

k	n	r	MHL_k/r	k	n	r	MHL_k/r
2	$(2^{26}, 2^{27})$	10^5	$(1.24, 1.24)$	3	$(3^{16}, 3^{17})$	10^5	$(1.48, 1.48)$
		10^6	$(1.23, 1.24)$			10^6	$(1.43, 1.46)$
		10^7	$(1.23, 1.24)$			10^7	$(1.00, 1.31)$
4	$(4^{13}, 4^{14})$	10^5	$(1.49, 1.50)$	5	$(5^{11}, 5^{12})$	10^5	$(1.47, 1.48)$
		10^6	$(1.45, 1.49)$			10^6	$(1.40, 1.46)$
		10^7	$(1.08, 1.38)$			10^7	$(0.83, 1.32)$
6	$(6^{10}, 6^{11})$	10^5	$(1.45, 1.46)$	7	$(7^9, 7^{10})$	10^5	$(1.42, 1.43)$
		10^6	$(1.38, 1.45)$			10^6	$(1.30, 1.42)$
		10^7	$(0.82, 1.32)$			10^7	$(0.55, 1.24)$
8	$(8^8, 8^9)$	10^5	$(1.38, 1.42)$				
		10^6	$(1.05, 1.37)$				
		10^7	$(0.21, 0.97)$				

Algorithm To Compute The Expected Header Length The result in Theorem 5 can be converted into an algorithm to compute the expected header length. The algorithm takes as input the values of k, n and r. Here $n = k^{\ell_0}$ for some $\ell_0 \geq 1$ and $1 \leq r \leq n$. The algorithm computes the values of $\gamma_{\ell,c}$ for each level ℓ in the tree $\mathcal{T}^0$ and each value of c. For fixed values of ℓ and c, computing $\gamma_{\ell,c}$ requires computing a fixed number of $\eta_r(\cdot,\cdot)$. One computation of $\eta_r(\cdot,\cdot)$ requires $O(r)$ multiplications. Hence, computing $\gamma_{\ell,c}$ also requires $O(r)$ multiplications. Since there are $\log_k n + 1$ levels in the tree, hence computing the expected header length requires $O(r \log n)$ multiplications. The algorithm requires constant amount of space. Hence, we have an algorithm requiring $O(r \log n)$ time and $O(1)$ space to compute the expected header length in the k-ary tree SD scheme for given values of k, n and r.

We have implemented the algorithm. Table 1 provides examples of outputs of the algorithm for different values of k, n and r.

5 Tackling Arbitrary Number Of Users

In the description of the scheme so far, we have assumed that n is a power of k. This may turn out to be restrictive in practice. Here we describe how to modify the scheme so as to be able to handle arbitrary number of users. When n is a power of k, the underlying structure is a *full* k-ary tree. In the more general case where n is not a power of k, we work with a *complete* k-ary tree. This is an analogue of complete binary trees used in data structures to describe heap algorithms.

The structure of a complete k-ary tree can be described as follows. Let $\ell_0 = \lceil \log_k n \rceil$ and $k^{\ell_0 - 1} < n \leq k^{\ell_0}$. By an abuse of notation, we denote by $\mathcal{T}^0$ the complete k-ary tree with n leaf nodes. The leaf nodes are at levels 0 and 1. Suppose that there are n_1 leaf nodes at level 0 and n_2 leaf nodes are at level 1. Let $n = k^{\ell_0 - 1} + i$ with $1 \leq i \leq k^{\ell_0} - k^{\ell_0 - 1}$. Then a simple calculation shows that $n_2 = k^{\ell_0 - 1} - \lceil i/(k-1) \rceil$ and $n_1 = n - n_2$. In $\mathcal{T}^0$, consider the path joining the root node 0 to the right-most internal node at level 1. Clearly, all subtrees rooted at nodes that are *not* on this path are full k-ary trees. In particular, subtrees rooted at nodes at level ℓ that are to the left (respectively right) of this path are of height ℓ (respectively $\ell - 1$). This path is consequently called the *dividing path*. If n is not a power of k, then subtrees rooted on the dividing path may not be full k-ary trees.

Definition Of The Collection $\mathcal{S}$ This remains unchanged, i.e., $\mathcal{S}$ still consists of $\mathcal{N}$ and subsets $S_{i,J}$ where i is an internal node in $\mathcal{T}^0$ and J is a subset of nodes with a common parent in the subtree of $\mathcal{T}^0$ rooted at i. The method for assigning keys to the subsets also remain unchanged. (In Section 6 later, we provide a different method for assigning keys.)

User Storage The actual number of seeds that a user will require depends on whether the user corresponds to a leaf at level 0 or a leaf at level 1. This number is at least $(2^{k-1}-1)\frac{\ell_0(\ell_0-1)}{2}$ and at most $(2^{k-1}-1)\frac{\ell_0(\ell_0+1)}{2}$. All users are attached to some node of the dividing path. Users to the left of the dividing path and attached to it at a level greater than 1, get $(2^{k-1}-1)\frac{\ell_0(\ell_0+1)}{2}$ (maximum number of) seeds; users to the right of the dividing path and attached to it at a level greater than 1, get $(2^{k-1}-1)\frac{\ell_0(\ell_0-1)}{2}$ (minimum number of) seeds. The number of seeds assigned to users attached to the dividing path at level 1 (to the rightmost internal node), is in the above mentioned range and can be easily calculated based upon the number of children of the last two nodes of the dividing path.

Cover Generation Algorithm The algorithm remains by and large the same. It is only at the initial stage that some modification is required. The cover generation algorithm for full k-ary trees progresses by processing the nodes in the list $\mathcal{L}$ one by one. This list is maintained as a queue and is initialized by inserting all the revoked users into it in the left-to-right order. All the users and so all the revoked users are necessarily at level 0.

In the case of complete trees, some of the users may be at level 1. So, the initialization of $\mathcal{L}$ is done by inserting all the revoked nodes at level 0 in the left-to-right order. At this point, the users at level 1 are not inserted into the list. The nodes in $\mathcal{L}$ are now processed one-by-one as in the cover generation algorithm. As part of this processing, the parents of these nodes get appended to $\mathcal{L}$. These parents are (internal) nodes at level 1. When the processing of the last revoked node at level 0 which is in $\mathcal{L}$ is completed, all nodes at level 1 which have at least one revoked child have been added to $\mathcal{L}$ in the left-to-right order. Now, all nodes corresponding to revoked users at level 1 are inserted into $\mathcal{L}$. From this point onwards there is no further change in the cover generation algorithm. It proceeds exactly as in the case of full trees and generates the cover. It is not difficult to argue that the algorithm correctly generates the cover. We have implemented this cover generation algorithm and used it in the analysis of average header length.

Header Length Analysis Moving from a full to a complete k-ary tree does not affect the upper bound on the header length of the algorithm. It is $\min(2r-1, n-r, \lceil n/k \rceil)$. For expected header length, as in the case of full k-ary trees, in theory, it is possible to develop an algorithm to compute the expected header length for the complete k-ary tree SD scheme.

As before let $X^i_{n,r}$ be the binary valued random variable which takes the value 1 if and only if the node i gives rise to a subset in the header. Hence, $\Pr[X^i_{n,r}=1]$ has to be computed using (16) and (17). To that end, the number of nodes under the subtree rooted at a node i has to be calculated and substituted appropriately in the equations. Note that the subtrees rooted at nodes on the dividing path may or may not be full. Thus, in order to compute $\Pr[X^i_{n,r}=1]$ for a node i using (16) and (17) it is required to consider a large number of cases depending on the relative position of a node with respect to the dividing path. While this can be done, the resulting algorithm becomes quite complicated and becomes difficult to implement.

In view of this difficulty, we have chosen not to implement the exact algorithm for finding the expected header length for complete k-ary trees. Instead, we have opted for a simulation study of the expected header length. For given values of k, n and r, we generate random revocation patterns using Floyd's algorithm [BF87] to sample r users from the set of n users. For each such random revocation pattern, the cover generation algorithm finds the exact cover and hence we get the header length for a particular revocation pattern. Taking the average of the header lengths obtained on different runs gives a statistical estimate of the expected header length. The number of iterations is chosen so that the average value of the header length stabilizes.

We have implemented this method. The result has been checked for accuracy in the following manner. In the case when n is a power of k, we have developed and implemented the algorithm to find the actual value of the expected header length. For such values of n, the results of the simulation study has been compared to the output of the exact algorithm and has been found to have tallied very well. Later, we report comparative performance analysis based on the simulation study of the expected header length.

6 Reducing User Storage

Given a $k \geq 2$, let n (not necessarily a power of k) be the number of users. By $\mathsf{us}_k(n)$ we denote the maximum number of m-bit seeds required to be stored by any of the n users in the system such that a user is able to generate the key associated to any subset in $\mathcal{S}$ of which it is a member. From (2), it appears that $\mathsf{us}_k(n)$ is $1 + (2^{k-1} - 1)\ell_0(\ell_0 + 1)/2$ where $\ell_0 = \lceil \log_k n \rceil$. In comparison to the case $k = 2$, for $k > 2$, the factor $(2^{k-1} - 1)$ contributes to the blow up in the key size. In this section, we describe methods by which this blow-up can be somewhat mitigated leading to values of us_k which are lower than that given by (2). For small values of k, in comparison to (2), the decrease in user storage that is attained is significant.

The reduction in user storage described in this section is achieved by deriving the seeds in a different fashion. The collection $\mathcal{S}$ remains unaltered and so the cover generation algorithm does not change. Also, the header length analysis (both maximum and expected), remains unaltered.

6.1 The Case $k = 3$

To explain the basic idea, we start by considering the case of $k = 3$. In this case, from (2) the maximum number of seeds required to be stored by any user is $1 + 3\ell_0(\ell_0 + 1)/2$, where $\ell_0 = \lceil \log_3 n \rceil$. We show that this can be reduced to $1 + \ell_0(\ell_0 + 1)$.

Consider the tree $\mathcal{T}^0$ where each internal node has (at most) 3 children. (For ease of understanding, one may initially assume $\mathcal{T}^0$ to be a full 3-ary tree.) Let j be an internal node of $\mathcal{T}^0$ and its children are nodes numbered $3j + 1, 3j + 2, 3j + 3$. Users in $\mathcal{T}^j$ get seeds derived from the seeds associated to j. There are two kinds of seeds associated to j: the uniform random seed L_j and the derived seed $L_{i,\{j\}}$ where i is some ancestor of j. For any such seed L, there are seven seeds $L_\sigma = G_\sigma(L)$, $0 \leq \sigma \leq 6$ which are derived from L. If L is of the form $L_{i,\{j\}}$, then L_0 is the key associated to the subset $S_{i,\{j\}}$, while all the other L_σ's are distributed to the users in $\mathcal{T}^j$ in the following manner. (We identify σ with their 3-bit binary representations.)

Users in $\mathcal{T}^{3j+1}$ get: $L_{011}, L_{010}, L_{001}$.
Users in $\mathcal{T}^{3j+2}$ get: $L_{101}, L_{100}, L_{001}$.
Users in $\mathcal{T}^{3j+3}$ get: $L_{110}, L_{100}, L_{010}$.

Hence, corresponding to the label L associated to j, each user in $\mathcal{T}^j$ gets three seeds. We show that by adopting a different strategy for generating the L_σ's, it is possible to provide each user in $\mathcal{T}^j$ with two seeds, from which it can generate the three required seeds.

The idea is based on replacing G by another cryptographic hash function $H : \{0, 1, 2\} \times \{0, 1\}^m \to \{0, 1\}^m$. For $b = 0, 1, 2$, we denote $H(b, seed)$ as $H_b(seed)$. We define $G_0(seed)$ to be $H_2(seed)$. Suppose σ is a t-bit string $b_1 \cdots b_t$. Then H_σ is defined to be $H_{b_t}(\cdots(H_{b_1}(seed)\cdots)$.

Consider again a seed L associated with the internal node j from which seeds for users in $\mathcal{T}^j$ are to be derived. For a t-bit binary string σ with $t \geq 1$, define $\widehat{L}_\sigma$ to be equal to $H_\sigma(L)$. A simple way of viewing this is the following. Consider an auxiliary full binary tree structure (independent of $\mathcal{T}^0$) of height t. Each path from the root to a leaf in this tree is of length t and is encoded as follows. Moving to the left child from a node is encoded by 0 and moving to the right child is encoded by 1. Then any node in the tree is encoded by a binary string σ which represents the path from the root to that node. The seed L is associated to the root node of the auxiliary

tree. The action of H on the seed of a node to derive the seeds of its children, results in the association of the seed $\widehat{L}_\sigma$ to the node encoded by σ.

The different L's to be distributed to the users in $\mathcal{T}^j$ are defined from the $\widehat{L}$'s by a suitable permutation on the set of all 3-bit strings. More concretely, we define

$$L_{001} = \widehat{L}_{000},\ L_{011} = \widehat{L}_{001},\ L_{010} = \widehat{L}_{010},\ L_{110} = \widehat{L}_{011},$$
$$L_{100} = \widehat{L}_{100},\ L_{101} = \widehat{L}_{101},\ L_{000} = \widehat{L}_{110},\ L_{111} = \widehat{L}_{111}.$$

The new assignment of seeds to users in $\mathcal{T}^j$ is as follows:

Users in $\mathcal{T}^{3j+1}$ get: $\widehat{L}_{00}, \widehat{L}_{010}$.
Users in $\mathcal{T}^{3j+2}$ get: $\widehat{L}_{10}, \widehat{L}_{000}$.
Users in $\mathcal{T}^{3j+3}$ get: $\widehat{L}_{01}, \widehat{L}_{100}$.

Since $L_{001} = \widehat{L}_{000} = H_0(\widehat{L}_{00})$, $L_{011} = \widehat{L}_{001} = H_1(\widehat{L}_{00})$ and $L_{010} = \widehat{L}_{010}$, users in $\mathcal{T}^{3j+1}$ can generate the required seeds. Similarly, the users in $\mathcal{T}^{3j+2}$ and $\mathcal{T}^{3j+3}$ can generate the seeds required by them.

The above method shows that for any seed associated to any internal node j, the number of derived seeds to be stored by users in $\mathcal{T}^j$ reduces to 2 from 3. As a result, the number of seeds required to be stored by any user is (at most) $1 + 2 \times \ell_0(\ell_0+1)/2 = 1 + \ell_0(\ell_0+1)$. This is summarized in the following result.

Proposition 6. *Suppose $k = 3$ and there are n users with $\ell_0 = \lceil \log_3 n \rceil$. Then the maximum number of m-bit seeds required to be stored by any user is $\mathsf{us}_3(n) = 1 + \ell_0(\ell_0+1)$.*

Given n, the expressions for us_2 and us_3 are as follows:

$$\mathsf{us}_2(n) = 1 + \frac{1}{2} \times \lceil \log_2 n \rceil \left(\lceil \log_2 n \rceil + 1 \right) \quad \text{and}$$
$$\mathsf{us}_3(n) = 1 + \lceil \log_3 n \rceil \left(\lceil \log_3 n \rceil + 1 \right). \tag{21}$$

It is interesting to form a comparative study of us_2 and us_3. This is done using the following sequence of results.

Lemma 7. *Let $\ell \geq 1$. Let s be the least positive integer such that $2^s > 3^\ell$. Then $3^\ell + 1 \leq 2^s < 2^{s+2} < 3^{\ell+2} + 1$. In other words, there are at least three powers of two between $3^\ell + 1$ and $3^{\ell+2} + 1$.*

Proof. Let $2^s = 3^\ell + x$ for some $x \geq 1$. Since s is the least positive integer such that $2^s > 3^\ell$, it follows that $2^{s-1} < 3^\ell$. From this, we get $3^\ell/2 + x/2 < 3^\ell$ so that $x < 3^\ell$. Now, $2^{s+2} < 3^{\ell+2}$ if $4 \times 3^\ell + 4x < 3^{\ell+2}$ if $x < (5/4)3^\ell$. Since we already have $x < 3^\ell$, the result follows. □

Lemma 8. *Let ℓ be a positive integer and s be the least positive integer such that $2^s > 3^\ell + 1$. Then the following holds.*

1. *If ℓ is even then $3s \geq 4(\ell+1)$. Further, the inequality is strict for even $\ell \geq 4$.*
2. *If $\ell \geq 5$ is odd then $3s \geq 4(\ell+1)$. Further, the inequality is strict for odd $\ell \geq 7$.*

Proof. We prove (1), the proof of (2) being similar. The proof is by induction on even $\ell \geq 2$. The base case is for $\ell = 2$ and then $s = 4$ and so the result holds. For the induction step, we first note that by Lemma 7, there are at least 3 powers of 2 between $3^\ell + 1$ and $3^{\ell+2} + 1$. So the least power of 2 which is greater than $3^{\ell+2} + 1$ is at least 2^{s+3}. By induction hypothesis, we have $3s > 4(\ell+1)$ and so $3(s+3) = 3s + 9 > 4(\ell+1) + 8 = 4(\ell+3)$. This shows the induction step. For $\ell = 4$, the inequality is strict and by the induction step, it follows that the inequality is strict for all even $\ell \geq 4$. □

Lemma 9. *Let $\ell \geq 4$ and s be the least positive integer such that $2^s > 3^\ell + 1$. Then for any n with $2^s + 1 \leq n \leq 3^{\ell+1}$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$.*

Proof. From the range of n it follows that $\mathsf{us}_3(n) = (\ell+1)(\ell+2)$. In the given range for n, $\mathsf{us}_2(n) \geq (s+1)(s+2)/2$.

We first prove the result by induction on even $\ell \geq 4$. For $\ell = 4$, $s = 7$ and the result holds. For the induction step, suppose the result holds for ℓ, i.e., $(s+1)(s+2)/2 > (\ell+1)(\ell+2)$. Also, by Lemma 8, we have $3s \geq 4(\ell+1)$. Consider the case for $\ell+2$. By Lemma 7, the least power of 2 which is greater than $3^{\ell+2}+1$ is at least 2^{s+3} and we have to consider n in the range $2^{s+3}+1 \leq n \leq 3^{\ell+3}$. In this range $\mathsf{us}_3(n) = (\ell+3)(\ell+4)$ and $\mathsf{us}_2(n) = (s+4)(s+5)/2$. The following computation shows the inductive step.

$$\begin{aligned}(s+4)(s+5)/2 &= 3s + 6 + (s+1)(s+2)/2 \\ &> 4(\ell+1) + 6 + (\ell+1)(\ell+2) = (\ell+3)(\ell+4).\end{aligned}$$

A similar argument by induction on odd $\ell \geq 5$ shows the result. □

Lemma 10. *Let $\ell \geq 7$ and s be the least positive integer such that $2^s > 3^\ell+1$. Then for any n with $3^\ell+1 \leq n \leq 2^s$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$.*

Proof. In the given range, $\mathsf{us}_3(n) = (\ell+2)(\ell+3)$ and $\mathsf{us}_2(n) = s(s+1)/2$. The induction is by separate induction for odd $\ell \geq 7$ (with corresponding $s = 12$) and even $\ell \geq 8$ (with corresponding $s = 13$). The base cases can be directly verified. The separate induction steps follow by an argument similar to that for Lemma 9. □

We finally get the following result.

Proposition 11. *Define I to be the following set of integers.*

$$I = \{3\} \cup [2^2+1, 3^2] \cup [2^4+1, 3^3] \cup [2^5+1, 3^4] \cup [2^7+1, 3^5] \cup [2^8+1, 3^6] \cup [2^{10}+1, \infty].$$

For $n \in I$, $\mathsf{us}_2(n) > \mathsf{us}_3(n)$ and for $n \in \mathbb{Z} \setminus I$, $\mathsf{us}_2(n) < \mathsf{us}_3(n)$.

Proof. For $n \leq 1024$, the result can be seen by direct computations. (Some of the cases also follow from Lemma 9.) For $n > 1024$, the combined effect of Lemma 9 and Lemma 10 shows the result. □

The above provides the complete comparison of the user storages for $k = 2$ and $k = 3$ and precisely proves that for $n > 1024$ the user storage required by the ternary tree based scheme is smaller than the user storage required by the binary tree based scheme. A similar observation with lesser refinement and without proof was made in [FKTS08].

6.2 The Case $k = 4$

As in the case for $k = 3$, let j be an internal node and L be a seed associated with j. Users in $\mathcal{T}^j$ obtain seeds L_σ derived from L using G_σ where in this case σ is a 4-bit string. More precisely, users in $\mathcal{T}^{4j+b}$, $1 \leq b \leq 4$, get seeds L_σ such that σ is a non-zero 4-bit string whose bth position from the left is zero. So, for the label L, each user in $\mathcal{T}^j$ gets 7 seeds.

In a manner similar to that of $k = 3$, it is possible to provide each user in $\mathcal{T}^j$ with 4 seeds such that from these seeds all the required 7 seeds can be derived. The idea is again based on using the function H to define certain seeds $\widehat{L}$'s and then define the L's in terms of the $\widehat{L}$'s. The definition of the $\widehat{L}$'s using H is the same as that in the case for $k = 3$. So, all we need to provide is the definition of L's in terms of the $\widehat{L}$'s. This is done as follows:

$L_{0111} = \widehat{L}_{0000}$, $L_{0110} = \widehat{L}_{0001}$, $L_{0100} = \widehat{L}_{0010}$, $L_{0101} = \widehat{L}_{0011}$,
$L_{1011} = \widehat{L}_{0100}$, $L_{0011} = \widehat{L}_{0101}$, $L_{0010} = \widehat{L}_{0110}$, $L_{1101} = \widehat{L}_{1001}$,
$L_{1001} = \widehat{L}_{1010}$, $L_{0001} = \widehat{L}_{1011}$, $L_{1110} = \widehat{L}_{1100}$, $L_{1100} = \widehat{L}_{1101}$,
$L_{1000} = \widehat{L}_{1110}$, $L_{1010} = \widehat{L}_{1111}$.

The distribution of seeds to the users in $\mathcal{T}^j$ is the following.

Users in $\mathcal{T}^{4j+1}$ get: $\widehat{L}_{00}, \widehat{L}_{0110}, \widehat{L}_{0101}, \widehat{L}_{1011}$.
Users in $\mathcal{T}^{4j+2}$ get: $\widehat{L}_{01}, \widehat{L}_{101}, \widehat{L}_{111}$.
Users in $\mathcal{T}^{4j+3}$ get: $\widehat{L}_{001}, \widehat{L}_{10}, \widehat{L}_{1101}, \widehat{L}_{1110}$.
Users in $\mathcal{T}^{4j+4}$ get: $\widehat{L}_{0001}, \widehat{L}_{0010}, \widehat{L}_{0110}, \widehat{L}_{11}$.

Using these seeds, each user can create the L's that it is supposed to get. For example, users in $\mathcal{T}^{4j+1}$ should be able to create

$$L_{0001}, L_{0010}, L_{0011}, L_{0100}, L_{0101}, L_{0110}, L_{0111}.$$

These can be obtained from the seeds obtained by the users in $\mathcal{T}^{4j+1}$ in the following manner.

$L_{0111} = \widehat{L}_{0000} = H_0(H_0(\widehat{L}_{00}))$; $L_{0110} = \widehat{L}_{0001} = H_1(H_0(\widehat{L}_{00}))$;
$L_{0100} = \widehat{L}_{0010} = H_0(H_1(\widehat{L}_{00}))$; $L_{0101} = \widehat{L}_{0011} = H_1(H_1(\widehat{L}_{00}))$;
$L_{0011} = \widehat{L}_{0101}$; $L_{0010} = \widehat{L}_{0110}$; $L_{0001} = \widehat{L}_{1011}$.

In a similar manner, users in the other subtrees of $\mathcal{T}^j$ can create the seeds required by them. Corresponding to the seed L associated with node j, the number of seeds to be stored by users in the subtree $\mathcal{T}^{4j+2}$ is 3, while users in all the other subtrees require to store 4 seeds. From this we get the following result.

Proposition 12. *Suppose $k = 4$ and there are n users with $\ell_0 = \lceil \log_4 n \rceil$. Then the maximum number of m-bit seeds required to be stored by any user is $\mathsf{us}_4(n) = 1 + 2\ell_0(\ell_0 + 1)$.*

Note that this is a significant improvement over the requirement of storing $1 + 3.5\ell_0(\ell_0 + 1)$ seeds as indicated by (2). The value of $\mathsf{us}_4(n)$, however, is greater than $\mathsf{us}_2(n)$ for $n \geq 4$. So, the user storage for binary trees is lesser than that for 4-ary trees.

6.3 The Technique For General k

Let $k \geq 3$ and consider the k-ary tree $\mathcal{T}^0$. As before, for any internal node j, there are two kinds of seeds associated with it: the uniform random label L_j and the label $L_{i,\{j\}}$ derived from some ancestor i of j. Let L be any such seed. Users in the tree $\mathcal{T}^j$ get seeds derived from L with users in the subtree $\mathcal{T}^{kj+b}$ getting all seeds $L_\sigma = G_\sigma(L)$ where σ is any non-zero k-bit string having a zero at position b from the left.

In the cases of $k = 3$ and $k = 4$, we have seen alternative ways of deriving L_σ. The idea has been to derive the $\widehat{L}_\sigma$'s using H and then define the L_σ's in terms of the $\widehat{L}_\sigma$'s. For the case of general k, it is still possible to derive the $\widehat{L}_\sigma$'s using H. The problem, however, is in defining the L's in terms of the $\widehat{L}_\sigma$'s. For $k = 3$ and $k = 4$, this has been done in a somewhat ad-hoc fashion and does not extend to the case for general k. Below we describe a more systematic method of deriving the L_σ's from L.

The idea is based on the notion of cyclotomic cosets. There are several equivalent ways of viewing cyclotomic cosets. The description that we give below is primarily based on cyclic shifts of bit strings. This is equivalent to the more conventional description [MS78] as we point out later.

Let σ be a k-bit string. Then the cyclotomic coset containing σ is the set of all k-bit strings that can be obtained by one or more circular left shifts of σ. Clearly there can be at most k elements in any cyclotomic coset and further, the number of elements in a cyclotomic coset is necessarily a divisor of k. So, if k is a prime, then the number of elements in any cyclotomic coset is either 1 or k. The all-zero string forms a cyclotomic coset by itself as does the all-one string. These are the only two cyclotomic cosets consisting of single elements. Given k, let χ_k denote the total number of cyclotomic cosets defined from k-bit strings.

Table 2: Examples of $M^{(k)}$ for $k = 3$, $k = 4$ and $k = 5$.

$k = 3$		
001	010	100
011	110	101

$k = 4$			
0001	0010	0100	1000
0011	0110	1100	1001
0101	1010		
0111	1110	1101	1011

$k = 5$				
00001	00010	00100	01000	10000
00011	00110	01100	11000	10001
00101	01010	10100	01001	10010
00111	01110	11100	11001	10011
01011	10110	01101	11010	10101
01111	11110	11101	11011	10111

The above can be described in terms of modulo arithmetic as follows. Let s be an integer in $[0, 2^k - 2]$. Then $2s \bmod 2^k - 1$ corresponds to a cyclic left shift of the k-bit binary representation of s. So, the cyclotomic coset containing the k-bit binary representation of s is essentially also the set of integers $s, 2s \bmod 2^k - 1, \ldots$.

If α is a generator of the field $GF(2^k)$, then α raised to the powers of elements (seen as integers) of one cyclotomic coset form the roots of one irreducible polynomial. Using this correspondence, the number $I(m)$ of irreducible polynomials of degree m over $GF(2)$ is given as [MS78]

$$I(m) = \frac{1}{m} \sum_{d|m} \mu(d) 2^{m/d} \tag{22}$$

where $\mu()$ is the Mobius function. The factorization of $x^{2^k} - x$ consists of all irreducible polynomials whose degrees divide k. The number of such polynomials is the number χ_k of cyclotomic cosets of k-bit strings and is obtained by summing $I(m)$ over all m which divides k. Using elementary results on the Mobius function, this turns out to be the following expression

$$\chi_k = \frac{1}{k} \sum_{t|k} \phi(t) 2^{k/t} \tag{23}$$

where $\phi()$ is the Euler totient.

Given two binary strings σ and τ of the same length, we define $\sigma \prec \tau$ if the integer represented by σ is less than the one represented by τ. In the following, we will assume that the elements of any cyclotomic coset are ordered from left-to-right based on $\prec$ and the first element will be called the coset representative. Further, we assume that the cyclotomic cosets are themselves ordered based on their coset representatives.

Let $C_0, \ldots, C_{\chi_k - 1}$ be the ordering of the cyclotomic cosets. Then C_0 is the coset containing the all-zero string and $C_{\chi_k - 1}$ is the coset containing the all-one string. We will consider only the cosets $C_1, \ldots, C_{\chi_k - 2}$. These are ordered in a matrix fashion with the ith row of the matrix consisting of the elements of C_i. Examples of the matrix for $k = 3$, $k = 4$ and $k = 5$ are given in Table 2. If k is prime, each row of the matrix will have k strings and if k is composite, the number of elements in the rows will be divisors of k.

Let us denote the matrix for k by $M^{(k)}$. Let the columns of $M^{(k)}$ be denoted by $V_1^{(k)}, \ldots, V_k^{(k)}$. Note that if k is composite, some of the $V_b^{(k)}$ will have blanks (the empty string) in their components. The non-empty strings in any column $V_b^{(k)}$ are obtained by a circular left shift of the corresponding elements in the column $V_{b-1}^{(k)}$. Extending this, the non-empty strings in $V_b^{(k)}$ are obtained as circular left shifts by b places of the corresponding elements in the column $V_1^{(k)}$. By construction, the first bit of each entry of $V_1^{(k)}$ is 0. By the left shift property, the bth bit position of each non-empty string in $V_b^{(k)}$ is 0.

Based on the matrix $M^{(k)}$ we define an auxiliary tree $T^{(k)}$. This tree is not a sub-tree of $\mathcal{T}^0$. (Note the difference in the notation between $\mathcal{T}^0$ and $T^{(k)}$.) Its role is to define the L_σ's in a manner such that the number of seeds required to be stored by a user reduces from the number $(2^{k-1} - 1)$ given by (2). There are a total of k levels in $T^{(k)}$ with the root node \$ at level 0 and the level numbers increasing as we move down the tree. The root node node \$ of $T^{(k)}$ has k children. By $T_b^{(k)}$, $b = 1, \ldots, k$, we denote the k subtrees rooted at these k nodes.

Each $T_b^{(k)}$ is a binary tree having $k-1$ levels numbered 1 to $k-1$ from top to bottom. The number of leaf nodes in $T_b^{(k)}$ is the number of non-empty strings in the column $V_b^{(k)}$ of $M^{(k)}$. The root node of $T_b^{(k)}$ is labelled by $(b, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda}$ is the empty string. (For simplicity, we sometimes write b instead of $(b, \boldsymbol{\lambda})$.) The other nodes of $T_b^{(k)}$ are labelled by a pair (b, τ), where τ is a binary string which encodes the path from the root b of $T_b^{(k)}$ to the node. The tree $T_b^{(k)}$ is not balanced. The construction of $T_b^{(k)}$ based on $V_b^{(k)}$ is described as follows.

1. The bth bit of each non-empty string in $V_b^{(k)}$ is 0. This corresponds to the root node $(b, \boldsymbol{\lambda})$ of $T_b^{(k)}$ at level numbered 1. Starting from the bth bit, we cyclically move right over the bit positions in the non-empty strings of $V_b^{(k)}$. Apart from bit position b, there are $k-1$ other positions in the non-empty strings in $V_b^{(k)}$. To these positions correspond the levels numbered 2 to $k-1$ of $T_b^{(k)}$.

2. There are two nodes at level numbered 2 of $T_b^{(k)}$ and these are labelled as $(b, 0)$ and $(b, 1)$. These nodes have binary trees rooted at them. All strings in $V_b^{(k)}$ whose $(b+1)$st bit position is 0 form the leaf nodes of the tree rooted at $(b, 0)$. Similarly, all strings in $V_b^{(k)}$ whose $(b+1)$st bit position is 1 form the leaf nodes of the tree rooted at $(b, 1)$.

3. Continuing the above, suppose the tree $T_b^{(k)}$ has been constructed up to level $l < k-1$. To construct the nodes at level $l+1$, we look at the $(b+l+1)$th bit position (cyclically from the right) of the strings in $V_b^{(k)}$. Let (b, τ) be a node at level l of $T_b^{(k)}$, so that τ is an $(l-1)$-bit string. Then 0τ is a substring in bit positions b to $b+l$ in one of the strings in $V_b^{(k)}$. Considering bit position $b+l+1$, the string 0τ is extended in two possible ways: $0\tau 0$ and $0\tau 1$. This gives rise to two children of (b, τ) labelled as $(b, \tau 0)$ and $(b, \tau 1)$.

As a consequence of this construction, to the leaf nodes of $T_b^{(k)}$ are associated the seeds L_σ where σ ranges over the non-empty strings in $V_b^{(k)}$. The top-to-bottom order in $V_b^{(k)}$ corresponds to the left-to-right order of the leaf nodes in $T_b^{(k)}$. The structure of $T^{(k)}$ for $k = 3, 4$ and 5 and the associated seeds L_σ's are shown in Figures 8, 9 and 10 respectively.

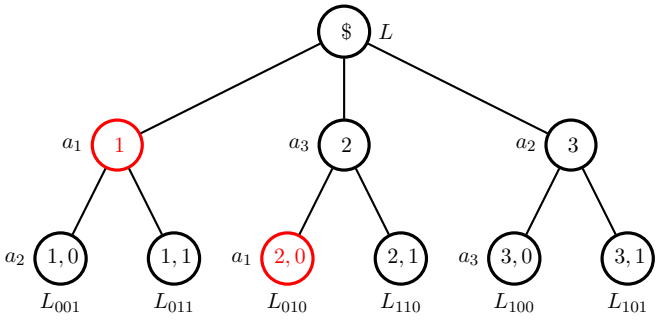

Figure 8: The structure of tree $T^{(3)}$. The seeds of the nodes marked with a_c are assigned to all users in $\mathcal{T}^{3j+c}$.

New key assignment: replacing G with F and H Consider again an internal node j of $\mathcal{T}^0$ and a seed L associated with the node j from which seeds L_σ's for the users in the subtree $\mathcal{T}^j$ are to be derived. The derivation of these seeds is done with the structure of $T^{(k)}$ and two hash functions $F : [1, k] \times \{0,1\}^m \to \{0,1\}^m$ and $H : \{0, 1, 2\} \times \{0,1\}^m \to \{0,1\}^m$. The function H is as used in Sections 6.1 and 6.2 while the function F is new. As before, we will use the notation $F_b(\cdot)$ and $H_c(\cdot)$ to denote the functions $F(b, \cdot)$ and $H(c, \cdot)$ respectively. For a binary string τ, the notation H_τ is as defined earlier.

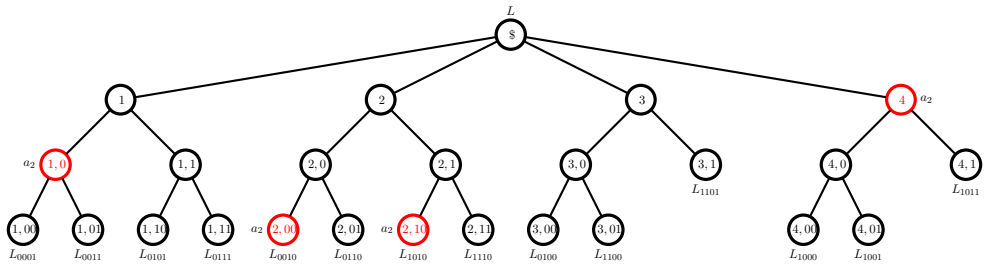

Figure 9: The structure of $T^{(4)}$.

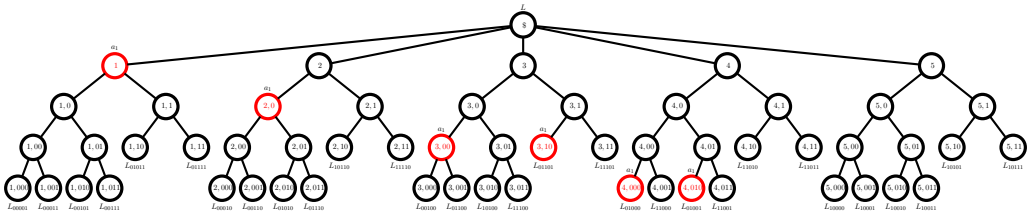

Figure 10: The structure of $T^{(5)}$.

The functions F and H together replace the function G used in Section 3.1 in the following manner. For any *seed*, the corresponding key is defined to be $H_2(seed)$ which in Section 3.1 was defined as $G_0(seed)$. The bth child of the root node of $T^{(k)}$ is given the seed $\widehat{L}_b = F_b(L)$. For any other node of $T_b^{(k)}$ labelled by a pair (b,τ), we associate the seed $\widehat{L}_{b,\tau} = H_\tau(F_b(L)) = H_\tau(L_b)$. Any leaf node of $T_b^{(k)}$ is labelled by a pair (b,τ) and has an associated L_σ. We define $L_\sigma = \widehat{L}_{b,\tau}$. This provides the definition of all the L_σ's that are required to be distributed to the users in the subtree $\mathcal{T}^j$.

We next look at the assignment of seeds to users. Each user in $\mathcal{T}^{kj+b}$ should be given a set of seeds such that it is able to generate all L_σ such that σ is a non-zero k-bit string whose bth position from the left is 0; also, it should not be able to generate any other seed. This is achieved by giving each user a subset of the $\widehat{L}$'s.

The seeds distributed to the users in $\mathcal{T}^{(kj+b)}$ are $\widehat{L}_{c,\tau}$ such that the following condition holds. In the subtree of $T^{(k)}$ rooted at the parent of the node labelled (c,τ) there is at least one leaf which is labelled by L_σ where the bth bit from the left in σ is 1.

For $k = 5$, the assignment is the following.

Users in $\mathcal{T}^{5j+1}$ get: $\widehat{L}_{1,\boldsymbol{\lambda}}$, $\widehat{L}_{2,0}$, $\widehat{L}_{3,00}$, $\widehat{L}_{3,10}$, $\widehat{L}_{4,000}$, $\widehat{L}_{4,010}$.
Users in $\mathcal{T}^{5j+2}$ get: $\widehat{L}_{5,\boldsymbol{\lambda}}$, $\widehat{L}_{1,0}$, $\widehat{L}_{2,00}$, $\widehat{L}_{2,10}$, $\widehat{L}_{3,000}$, $\widehat{L}_{3,010}$.
Users in $\mathcal{T}^{5j+3}$ get: $\widehat{L}_{4,\boldsymbol{\lambda}}$, $\widehat{L}_{5,0}$, $\widehat{L}_{1,00}$, $\widehat{L}_{1,10}$, $\widehat{L}_{2,000}$, $\widehat{L}_{2,010}$.
Users in $\mathcal{T}^{5j+4}$ get: $\widehat{L}_{3,\boldsymbol{\lambda}}$, $\widehat{L}_{4,0}$, $\widehat{L}_{5,00}$, $\widehat{L}_{5,10}$, $\widehat{L}_{1,000}$, $\widehat{L}_{1,010}$.
Users in $\mathcal{T}^{5j+5}$ get: $\widehat{L}_{2,\boldsymbol{\lambda}}$, $\widehat{L}_{3,0}$, $\widehat{L}_{4,00}$, $\widehat{L}_{4,10}$, $\widehat{L}_{5,000}$, $\widehat{L}_{5,010}$.

The total number of seeds assigned to any user is given by the following result.

Proposition 13. *Let $k \geq 3$, $n \geq 1$ and $\ell_0 = \lceil \log_k n \rceil$. Then $\mathsf{us}_k(n) = (\chi_k - 2)(\ell_0(\ell_0+1))/2$.*

Proof. Consider the tree $T^{(k)}$. The root node has k children $T_1^{(k)}, \ldots, T_k^{(k)}$. Seeds of the form $\widehat{L}_{b,\tau}$ associated with the nodes of these trees are assigned to the different users. The leaf nodes of $T_b^{(k)}$ are also labelled by the seeds L_σ's which are elements of $V_b^{(k)}$, the bth column of the matrix $M^{(k)}$. Recall that the non-empty strings in

Table 3: Reduction of user storage achieved by Proposition 13 in comparison to 2. In each case, $\ell_0 = \lceil \log_k n \rceil$.

	$k=3$	$k=4$	$k=5$
Eqn. (2)	$1 + 1.5\ell_0(\ell_0+1)$	$1 + 3.5\ell_0(\ell_0+1)$	$1 + 7.5\ell_0(\ell_0+1)$
us_k	$1 + \ell_0(\ell_0+1)$	$1 + 2\ell_0(\ell_0+1)$	$1 + 3\ell_0(\ell_0+1)$
	$k=6$	$k=7$	$k=8$
Eqn. (2)	$1 + 15.5\ell_0(\ell_0+1)$	$1 + 31.5\ell_0(\ell_0+1)$	$1 + 63.5\ell_0(\ell_0+1)$
us_k	$1 + 6\ell_0(\ell_0+1)$	$1 + 9\ell_0(\ell_0+1)$	$1 + 18.5\ell_0(\ell_0+1)$

the bth column of $M^{(k)}$ are obtained by a cyclic left shift of the corresponding strings in the $(b-1)$th column of $M^{(k)}$.

As a result, the σ's corresponding to the labels L_σ's of the leaf nodes of $T_b^{(k)}$ are obtained by a cyclic left shift of the respective ζ's corresponding to the labels L_ζ's of the leaf nodes of $T_{b-1}^{(k)}$. Due to this, the following symmetry property holds. For $b > 1$, if the users in $\mathcal{T}^{kj+b}$ get x seeds of the form $\widehat{L}_{1,\tau}$, then the users in $\mathcal{T}^{kj+1}$ get (at most) x seeds of the form $\widehat{L}_{b,\tau}$.

A consequence of this symmetry property is that the number of seeds given to the users in $\mathcal{T}^{kj+1}$ is equal to the number of seeds of the form $\widehat{L}_{1,\tau}$ which are assigned to all the users. By construction, if τ ends with a 0, then the corresponding $\widehat{L}_{1,\tau}$ is assigned to some user. So, the only seeds of the form $\widehat{L}_{1,\tau}$ which are not assigned to any user are those ending with a 1. These seeds correspond exactly to the nodes of $T_1^{(k)}$ which are the right children of some node.

The number of leaf nodes of $T_1^{(k)}$ is the number of strings in $V_1^{(k)}$ which in turn is equal to $\chi_k - 2$. Since $T_1^{(k)}$ is a binary tree, the number of internal nodes of $T_1^{(k)}$ is equal to $\chi_k - 3$. So, the total number of nodes of $T_1^{(k)}$ is $2\chi_k - 5$. Each internal node has exactly one child node and so the number of nodes which are right children is equal to the number of internal nodes of $T_1^{(k)}$ which is $\chi_k - 3$. As a result, the number of nodes of $T_1^{(k)}$ which are labelled by $(1, \tau)$ with τ ending with 0 is equal to $2\chi_k - 5 - (\chi_k - 3) = \chi_k - 2$.

□

Note that $\chi_3 = 4$ and $\chi_4 = 6$ and so the user storage given by this result agrees with the user storage given in Propositions 6 and 12 respectively. The methods of deriving the seeds, however, are different.

In Table 3, we provide a comparison of the user storage given by Proposition 13 to that given by (2). In concrete terms, the reduction is quite significant. Comparing to the user storage for $k = 2$, the increase is only a few times. This can be seen from the values of $\mathsf{us}_2(n)$ and $\mathsf{us}_k(n)$ for $k > 2$ and different values of n as given in Table 5.

7 The Layered k-ary Tree Subset Difference Scheme

The idea of layering the levels of the underlying binary tree $\mathcal{T}^0$ of the NNL-SD scheme in order to reduce the user storage was introduced in [HS02]. Here we apply the same technique to reduce the storage of the k-ary tree generalization of the SD scheme.

As before, we work with an underlying full k-ary tree with $n = k^{\ell_0}$ leaf nodes. Nodes at equal distances from the root are said to be at the same level. There are $\ell_0 = \log_k n$ levels in the tree $\mathcal{T}^0$. Some of these levels are marked as *special*. A *layer* is defined to be the levels in between and including two consecutive special levels. Hence, a layering strategy $\boldsymbol{\ell}$ is defined by the numbers of the special levels $\ell_0 > \ell_1 > \ldots > \ell_{e-1} > \ell_e = 0$.

Let $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{e-1}, \ell_e)$ be a layering strategy. There are $e+1$ special levels in $\boldsymbol{\ell}$. An alternate representation of the layering strategy is by the length of each layer. For $1 \leq i \leq e$, we define $d_i = \ell_{i-1} - \ell_i$ so that d_i's

are positive integers whose sum is ℓ_0. At the same time, given any sequence of positive integers $\boldsymbol{d} = (d_1, \ldots, d_e)$ whose sum is ℓ_0, it is possible to define a layering scheme where $\ell_i = \ell_0 - \sum_{j=1}^{i} d_j$.

The Collection $\mathcal{S}$ And Key Assignment Let j be an internal node in $\mathcal{T}^i$ having k children in $\mathcal{T}^0$ namely $\{kj+1, \ldots, kj+k\}$. Let $J \subset \{kj+1, \ldots, kj+k\}$ such that $0 < |J| < k$. Subsets in the collection $\mathcal{S}$ are of the form $S_{i,J}$ as has been described in Section 3.1 for the k-ary tree SD scheme. Each internal node i is assigned a uniform random seed L_i as before. However, unlike the k-ary tree SD scheme, not all subsets of the form $S_{i,J}$ are assigned keys. With the introduction of layering, only certain subsets of the form $S_{i,J}$ are assigned keys. These subsets are of the following type:

- If i is at a special level, then J can be any set of nodes that are siblings in the subtree $\mathcal{T}^i$.
- If i is not at a special level, then J will be a set of nodes that are siblings in the subtree $\mathcal{T}^i$ and in the same layer as i.

We have seen in Section 3 and Section 6 two different techniques to derive the key $L_{i,J}$ for a set $S_{i,J}$. (One may recollect here that the collection $\mathcal{S}$ of subsets remains unchanged for both key assignment techniques.) For the layered version of the scheme, we assume that the second technique of Section 6 that requires lesser user storage, is used to assign keys to subsets.

User Storage Given a layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{e-1}, \ell_e)$ in a tree with $n = k^{\ell_0}$ leaves, we compute the number of seeds that a user needs to store. For an ancestor i of the user that is at a special level ℓ, the user has to store $(\chi_k - 2)$ seeds derived by sets of nodes that are directly attached to (or "falling off from") the path between the user leaf and i. Hence the total number of seeds to be stored for an ancestor at a special level is $(\chi_k - 2)\ell$. Similarly, for an ancestor of the user that is at a non-special level ℓ which is between two special levels ℓ_{i-1} and ℓ_i $(\ell_{i-1} < \ell < \ell_i)$ the user has to store $(\chi_k - 2)(\ell - \ell_i)$ seeds. Hence the user storage for the layering strategy $\boldsymbol{\ell}$ is

$$\mathsf{storage}_0^k(\boldsymbol{\ell}) = (\chi_k - 2) \times \left(\sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=\ell_{i+1}+1}^{\ell_i - 1} (j - \ell_{i+1}) \right) \tag{24}$$

where $\ell_0 = \lceil \log_k n \rceil$. The expression to compute $\mathsf{storage}_0^k(\boldsymbol{\ell})$ in (24) for general k is derived by a similar logic as used in [BS14] for $k = 2$. The storage requirement derived for $k = 2$ from (24), is exactly the same as found in [BS14].

7.1 Storage Minimal Layering

Now, let us consider two extreme layering strategies and find their storage requirement. The first layering strategy has only the top-most and bottom-most levels as special and hence $\boldsymbol{\ell} = (\ell_0, 0)$. It can be easily seen that this scheme is the same as the k-ary tree SD scheme and hence has the same storage requirement as that of the k-ary tree SD scheme.

As more special levels are introduced in between these two levels, the user storage should go down. This is because, the number of seeds derived from the nodes at non-special levels above the bottom-most layer, reduces in the user storage.

However, we see that as we continue marking more and more levels as special, we finally get the layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_0 - 1, \ldots, 1, 0)$ where all the levels are marked as special. The resultant scheme is again exactly the same as the k-ary tree SD scheme. Hence, as for binary trees in [BS14], there should exist a layering strategy of the k-ary trees that results in minimum storage.

For a given k and ℓ_0, let $\text{SML}_0^k(\ell_0)$ denote a layering strategy $\boldsymbol{\ell}$ (or equivalently given by the sequence of differences $\boldsymbol{d}$), such that $\mathsf{storage}_0^k(\boldsymbol{\ell})$ takes the minimum value among all possible layering strategies. Let $\#\text{SML}_0^k(\ell_0)$ denote the storage requirement $\mathsf{storage}_0^k(\boldsymbol{\ell})$ for the storage minimal layering strategy $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_e)$.

The storage minimal layering strategy $\text{SML}_0^k(\ell_0)$ can be found using a dynamic programming algorithm as follows. We first fix the number of layers e in a layering strategy. Out of all the storage requirements of these layering strategies one will be minimum. Let $\text{SML}_0^k(e, \ell_0)$ denote a layering strategy that requires minimum storage amongst all layerings with e layers. The number of layers e can be at least 1 and at most ℓ_0. Hence, $\text{SML}_0^k(\ell_0)$ will be the minimum of all these layering strategies over all values of e. So we get

$$\#\text{SML}_0^k(\ell_0) = \min_{1 \le e \le \ell_0} \#\text{SML}_0^k(e, \ell_0). \tag{25}$$

Similarly, $\#\text{SML}_0^k(e, \ell_0)$ is the minimum storage requirement amongst all the layering strategies for a given number of layers e. So we get

$$\#\text{SML}_0^k(e, \ell_0) = \min_{(\ell_0, \ell_1, \ldots, \ell_e)} \mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e). \tag{26}$$

We write the expression to compute $\mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e)$ on the right hand side of (24) in a recursive fashion as follows

$$\begin{aligned} \mathsf{storage}_0^k(\ell_0, \ell_1, \ldots, \ell_e) &= (\chi_k - 2) \times \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} \right) \\ &\quad + \mathsf{storage}_0^k(\ell_1, \ldots, \ell_e). \end{aligned} \tag{27}$$

Using (27) and (26), we get a recursive definition of $\#\text{SML}_0^k(e, \ell_0)$ in terms of $\#\text{SML}_0^k(e-1, \ell_1)$ as follows

$$\begin{aligned} &\#\text{SML}_0^k(e, \ell_0) = \\ &\min_{1 \le \ell_1 < \ell_0} \left((\chi_k - 2) \times \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} \right) + \#\text{SML}_0^k(e-1, \ell_1) \right). \end{aligned} \tag{28}$$

This recursive definition of (28) is the basis for our dynamic programming algorithm. A similar dynamic programming algorithm to compute the $\text{SML}_0^2(\ell_0)$ for layering in binary trees has been proposed in [BS14]. The above algorithm is a generalization using k-ary trees of that algorithm using binary trees.

Empirical Analysis In Proposition 6.2 we have seen that the storage requirement of the k-ary tree SD scheme for $k = 3$ is lesser than that of $k = 2$ (the binary tree case) for $n \ge 2^{10}$. We know from [HS02, BS14] that the user storage of a subset difference based scheme can be reduced using different layering strategies. Hence, it is of interest to check the effect of layering on the k-ary tree SD scheme.

We have implemented the dynamic programming algorithm for finding the storage minimal layering in the k-ary tree SD scheme. Executing this algorithm for computing the storage minimal layering for $k = 3$ for different values of n and comparing with the case when $k = 2$, we find the range of n where the storage due to $k = 3$ is less than the storage due to $k = 2$. Table 4 lists those ranges for $n < 2^{30}$ and the corresponding storage requirements.

8 A Comparative Study

Table 5 provides a comparative study of the mean header length MHL_k and the user storage us_k as k varies from 2 to 8. For the study, we have varied n from 10^3 to 10^8. Since n is not a power of k, the complete tree extension

Range of n	$(\#\text{SML}_0^2, \#\text{SML}_0^3)$	Range of n	$(\#\text{SML}_0^2, \#\text{SML}_0^3)$
$\{2^4+1,\ldots,3^3\}$	$(11,10)$	$\{2^6+1,\ldots,3^4\}$	$(18,16)$
$\{2^7+1,\ldots,3^5\}$	$(22,22)$	$\{2^9+1,\ldots,3^6\}$	$(30,28)$
$\{2^{11}+1,\ldots,3^7\}$	$(40,36)$	$\{2^{12}+1,\ldots,3^8\}$	$(45,44)$
$\{2^{14}+1,\ldots,3^9\}$	$(55,52)$	$\{2^{15}+1,\ldots,3^{10}\}$	$(61,60)$
$\{2^{17}+1,\ldots,3^{11}\}$	$(73,70)$	$\{2^{19}+1,\ldots,3^{12}\}$	$(85,80)$
$\{2^{20}+1,\ldots,3^{13}\}$	$(91,90)$	$\{2^{22}+1,\ldots,3^{14}\}$	$(105,100)$
$\{2^{23}+1,\ldots,3^{15}\}$	$(112,110)$	$\{2^{25}+1,\ldots,3^{16}\}$	$(126,122)$
$\{2^{28}+1,\ldots,3^{18}\}$	$(148,146)$		

Table 4: Ranges of n ($< 2^{30}$) such that $\#\text{SML}_0^2 > \#\text{SML}_0^3$.

of the scheme described in Section 5 has been used. The reported results for MHL_k has been done using the simulation program. (Earlier, in Table 1 we have provided results based on running the algorithm for computing the expected header length when n is a power of k.) User storage is obtained from Proposition 13. We observe the following from Table 5:

- For small values of r/n, $\mathsf{MHL}_k/r > \mathsf{MHL}_2/r$ while for larger values of r/n, $\mathsf{MHL}_k/r < \mathsf{MHL}_2/r$. This indicates that for a given $k > 2$, there is a threshold value $\delta_k \in (0,1)$ such that for $r/n > \delta_k$, the mean header length of the k-ary tree SD scheme is smaller than that for $k = 2$.
- For a fixed k, the values of MHL_k/r are (almost) the same for a given ratio r/n for any arbitrary n. This behavior is captured in Table 6 and the corresponding plot of its data is given in Figure 11. The almost straight red line in Figure 11 shows the behavior for $k = 2$. For other values of k, the points where the respective curves intersect this straight line correspond to $r/n = \delta_k$. These approximate values of δ_k are shown in Table 7. We see that as k increases the value of δ_k decreases and consequently, the performance of the k-ary tree SD scheme is better than $k = 2$ for a larger range of values of r.

Practical Considerations An important application of broadcast encryption is pay-per-view of cable TV and DTH services. In cable TV systems, a set of basic channels are free to air and are not scrambled. Hence, everyone with a cable TV connection can view these channels. All other channels are encrypted. For paid channels or pay-per-view programmes, it is quite likely that the number of users subscribing to the channel/programme is substantially less than the total number of customers n of the cable company. Hence, the number of revoked users is of the magnitude of n.

As an example, in Table 5 consider a pay-TV application with $n = 10^8$ users. Supposing r to be about $0.4n$ and assuming 128-bit keys, the bandwidth savings of $k = 8$ over $k = 2$ for that channel/programme is 244MB per session. The user storage, on the other hand, increases from 5.9KB to 23.9KB. Due to steadily decreasing memory prices, the cumulative benefit of savings in communication bandwidth over a period of time is likely to outweigh the cost of extra memory.

For applications like content protection (DRM) in optical discs like HD-DVD and Blu-ray discs [AAC] the amount of space reserved in the discs for the header is fixed. Since the header length of the NNL-SD scheme is linear in the number of revoked users r of the system, it would be able to tolerate up to a fixed number of revoked users (say) r_{max}. The generalization using k-ary trees brings down the average header length in most cases (when $r/n > \delta_k$) and the worst case header length is at most as much as that for $k = 2$. Hence, the threshold r_{max} will be larger for higher values of k and so the system will be able to tolerate more revocations.

Table 5: User storage and mean header lengths in the complete k-ary tree scheme for values of k between 2 and 8. For a fixed n, we report MHL_k/r for three different choices of r namely, $r = (0.1n, 0.2n, 0.4n)$.

n	k	us_k	MHL_k/r	n	k	us_k	MHL_k/r
10^3	2	55	$(1.10, 0.98, 0.72)$	10^4	2	105	$(1.11, 0.97, 0.71)$
	3	56	$(1.27, 1.06, 0.72)$		3	90	$(1.26, 1.07, 0.72)$
	4	60	$(1.21, 0.96, 0.59)$		4	112	$(1.20, 0.96, 0.59)$
	5	90	$(1.11, 0.84, 0.50)$		5	126	$(1.11, 0.84, 0.49)$
	6	120	$(1.03, 0.73, 0.42)$		6	252	$(1.02, 0.73, 0.41)$
	7	180	$(0.95, 0.65, 0.36)$		7	270	$(0.94, 0.65, 0.36)$
	8	340	$(0.86, 0.58, 0.32)$		8	510	$(0.86, 0.58, 0.31)$
10^5	2	153	$(1.11, 0.97, 0.71)$	10^6	2	210	$(1.11, 0.97, 0.71)$
	3	132	$(1.27, 1.06, 0.72)$		3	182	$(1.27, 1.07, 0.72)$
	4	180	$(1.20, 0.96, 0.59)$		4	220	$(1.20, 0.96, 0.59)$
	5	216	$(1.11, 0.84, 0.49)$		5	270	$(1.11, 0.84, 0.49)$
	6	336	$(1.02, 0.73, 0.41)$		6	432	$(1.02, 0.73, 0.41)$
	7	378	$(0.94, 0.65, 0.36)$		7	648	$(0.94, 0.65, 0.36)$
	8	714	$(0.87, 0.58, 0.31)$		8	952	$(0.87, 0.58, 0.31)$
10^7	2	300	$(1.11, 0.97, 0.71)$	10^8	2	378	$(1.11, 0.97, 0.71)$
	3	240	$(1.27, 1.06, 0.72)$		3	306	$(1.27, 1.06, 0.72)$
	4	312	$(1.20, 0.96, 0.59)$		4	420	$(1.20, 0.96, 0.59)$
	5	396	$(1.11, 0.84, 0.49)$		5	468	$(1.11, 0.84, 0.49)$
	6	540	$(1.02, 0.73, 0.41)$		6	792	$(1.02, 0.73, 0.41)$
	7	810	$(0.94, 0.65, 0.36)$		7	990	$(0.94, 0.65, 0.36)$
	8	1224	$(0.87, 0.58, 0.31)$		8	1530	$(0.87, 0.58, 0.31)$

Table 6: List of values of the ratio MHL_k/r (for any n) corresponding to the varying ratio r/n for each k. For a given $k > 2$, the values in bold indicate the minimum value of r/n from where the scheme performs better than that for $k = 2$.

k \ r/n	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
2	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
3	(1.46,	1.37,	1.27,	1.06,	0.88,	0.72,	**0.57**,	0.43,	0.31,	0.20,	0.10,	0.00)
4	(1.47,	1.35,	1.20,	**0.96**,	0.76,	0.59,	0.47,	0.36,	0.27,	0.18,	0.10,	0.00)
5	(1.44,	1.28,	1.11,	**0.84**,	0.63,	0.49,	0.39,	0.31,	0.24,	0.17,	0.09,	0.00)
6	(1.41,	1.22,	**1.02**,	0.73,	0.54,	0.41,	0.33,	0.27,	0.21,	0.15,	0.09,	0.00)
7	(1.38,	**1.16**,	0.94,	0.65,	0.47,	0.36,	0.28,	0.23,	0.19,	0.14,	0.08,	0.00)
8	(1.34,	**1.11**,	0.87,	0.58,	0.41,	0.31,	0.25,	0.21,	0.17,	0.13,	0.08,	0.00)
16	(**1.22**,	0.78,	0.55,	0.31,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

Table 7: Values of the threshold δ_k.

k	3	4	5	6	7	8	16
δ_k	0.44	0.19	0.11	0.07	0.05	0.04	< 0.01

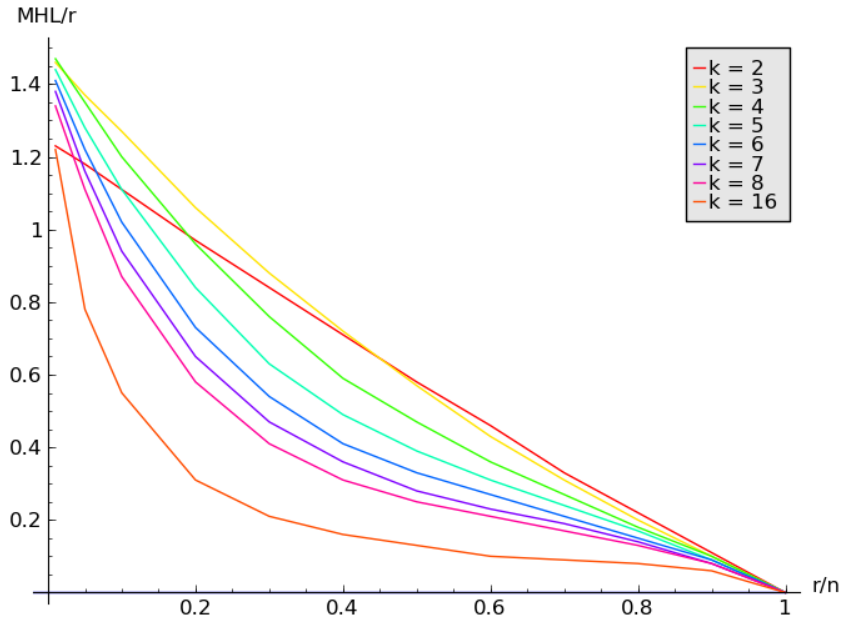

Figure 11: Plot showing how MHL_k/r varies with r/n.

9 Security Analysis

The security of the general Subset-Cover framework was proved in [NNL01, NNL02] in terms of *key indistinguishability*. Both the NNL-SD scheme and the k-ary tree scheme fall under the Subset-Cover framework. As such the security analysis performed in [NNL01, NNL02] also apply to the k-ary tree scheme. The only point that needs some justification is that the key indistinguishability property also holds for the k-ary tree scheme. Our purpose in this section is provide this justification. First, we briefly recall the Subset-Cover framework as described in [NNL01, NNL02]. It employs two cryptographic primitives $E_K^{(1)}$ and $E_L^{(2)}$.

- The function $E^{(1)} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ is used to encrypt the message M with a *session key* $K \in \mathcal{K}$; $E_K^{(1)}(\cdot) \triangleq E^{(1)}(K, \cdot)$ is length preserving. The session key is a random string chosen afresh for each new message M.
- The function $E^{(2)} : \mathcal{K}_1 \times \{0,1\}^m \to \{0,1\}^m$ is used to encrypt the session key K with a long-lived key $L \in \mathcal{K}_1$ corresponding to the subset S_j ($\in \mathcal{S}_c$) of users; $E_L^{(2)}(\cdot) \triangleq E^{(2)}(L, \cdot)$ is length preserving.

As discussed earlier, in order to broadcast a message M, the center chooses a random session key K and encrypts M as $E_K^{(1)}(M)$. This session key K has to be securely communicated to the privileged users in $\mathcal{N} \setminus \mathcal{R}$ so that they can correctly get back K and in turn obtain M from the encrypted body. To that end, the center finds the subset cover $\mathcal{S}_c = \{S_{i_1}, \ldots, S_{i_h}\} \subset \mathcal{S}$. Let $L_{i_1}, \ldots, L_{i_h}$ be the respective long-lived keys that were assigned to each of these subsets. The center then encrypts the session key K with each of these keys L_{i_j} as follows:

$$E_{L_{i_1}}^{(2)}(K), E_{L_{i_2}}^{(2)}(K), \ldots, E_{L_{i_h}}^{(2)}(K).$$

The h encryptions of the session key are sent along with $E_K^{(1)}(M)$ as the header for the encrypted message. The encrypted message $E_K^{(1)}(M)$ along with the header forms the ciphertext C.

The security analysis in [NNL01, NNL02] starts by assuming that the probability of distinguishing the encryption of a true message from that of a random message is negligible for both the primitives $E_K^{(1)}$ and $E_L^{(2)}$. With these assumptions, it was shown that if the key assignment technique of the subset cover algorithm satisfies the key indistinguishability property (a notion introduced in [NNL01, NNL02]), then one obtains a secure encryption of the message that cannot be distinguished by an adversary from the encryption of a random string of the same length as the message. The adversary is assumed to hold the keys of all subsets other than those for which the message has been encrypted.

Formal Definitions from [NNL01, NNL02] The security requirements of the two underlying primitives $E_K^{(1)}$ and $E_L^{(2)}$ are defined separately in [NNL01, NNL02]. Note that $E_K^{(1)}$ uses short-lived keys (only for a session) whereas $E_L^{(2)}$ uses long-lived ones (to be used for multiple sessions over the lifetime of the scheme). The assumptions on the security of these primitives are as follows.

An adversary $\mathcal{B}$ for $E^{(1)}$ chooses a message M and receives as input one of the following: (a) $E_K^{(1)}(M)$ or (b) $E_K^{(1)}(R_M)$ where R_M is a random message of the same length as M and K is a uniform random key. $\mathcal{B}$ outputs 'true' if it identifies its input as $E_K^{(1)}(M)$ and 'false' otherwise. The ability of $\mathcal{B}$ to be able to distinguish between these two encryptions is formalized as follows:

$$|\Pr[\mathcal{B} \text{ outputs 'true'}|E_K^{(1)}(M)] - \Pr[\mathcal{B} \text{ outputs 'true'}|E_K^{(1)}(R_M)]| \leq \epsilon_1.$$

For the primitive $E^{(2)}$, the definition is as follows: An adversary $\mathcal{B}$ gets to choose adaptively polynomially many messages and obtains ciphertexts encrypted with $E_L^{(2)}$ (where L is a uniform random key) and similarly provides ciphertexts and obtains the corresponding messages. Then, it chooses a random plaintext K and receives one of the following: (a) $E_L^{(2)}(K)$ or (b) $E_L^{(2)}(R_K)$ where R_K is a random string of length $|K|$. $\mathcal{B}$ outputs 'true' if it identifies its input as $E_L^{(2)}(K)$ and 'false' otherwise. The ability of $\mathcal{B}$ to be able to distinguish between these two encryptions is formalized as follows:

$$|\Pr[\mathcal{B} \text{ outputs 'true'}|E_L^{(2)}(K)] - \Pr[\mathcal{B} \text{ outputs 'true'}|E_L^{(2)}(R_K)]| \leq \epsilon_2.$$

Key Assignment As pointed out in [NNL01, NNL02], a secure subset cover algorithm requires the key assignment technique to have the key indistinguishability property. This property requires that the key L assigned to a subset $S \in \mathcal{S}$ is indistinguishable from a random key, given all the secret information of all users in $\mathcal{N} \setminus S$.

Definition 1. *[NNL01, NNL02] Let $\mathcal{A}$ be a subset cover revocation algorithm that defines the collection $\mathcal{S}$ of subsets of $\mathcal{N}$. Let $\mathcal{B}$ be a feasible adversary that selects a subset $S \in \mathcal{S}$ and then receives the I_u for all $u \in \mathcal{N} \setminus S$. Then $\mathcal{A}$ is said to satisfy the key indistinguishability property if the probability that $\mathcal{B}$ distinguishes the 'true' key L (the long-lived key of the set S that was chosen by $\mathcal{B}$) from a random key R_L of the same length $|L|$, is negligible and bounded above by ϵ_3. In other words,*

$$|\Pr[\mathcal{B} \textit{ outputs 'true'}|L] - \Pr[\mathcal{B} \textit{ outputs 'true'}|R_L]| \leq \epsilon_3.$$

The key assignment for a scheme in the Subset-Cover framework is *information theoretic* if the keys associated to the different subsets are chosen uniformly and independently at random. For information theoretic key assignment schemes, it can be easily seen that the key indistinguishability property of Definition 1 holds with $\epsilon_3 = 0$.

Definition 2. *[NNL01, NNL02] Consider an adversary $\mathcal{B}$ that gets to:*

1. *Adaptively select a set $\mathcal{R}$ of revoked users and obtain I_u for all $u \in \mathcal{R}$. By adaptively, it is meant that $\mathcal{B}$ may select messages $M_1, M_2, \ldots$ and revocation sets $\mathcal{R}_1, \mathcal{R}_2, \ldots$ and observe the encryption of M_i when the revoked set is $\mathcal{R}_i$. The users in $\mathcal{R}_i$ may or may not be corrupted. $\mathcal{B}$ can also create a ciphertext and see how any (non-corrupted) user decrypts it. It then asks to corrupt a receiver u and obtains I_u. This step is then repeated $|\mathcal{R}|$ times for any $u \in \mathcal{N}$.*

2. *Choose a message M as the challenge plaintext and a set $\mathcal{R}$ of revoked users that must include all the users it has corrupted at least (if not more).*

$\mathcal{B}$ then receives the encryption for a message M' and the revoked set $\mathcal{R}$. It has to guess whether M' is one of the following: (a) the message M it chose or (b) a random string R_M of length $|M|$.

The revocation scheme is said to be secure if for any (probabilistic polynomial time) adversary $\mathcal{B}$ as above, the probability that $\mathcal{B}$ distinguishes between the two cases (a) and (b) is negligible.

The Security Theorem The main security theorem below shows that the key indistinguishability property is sufficient for a scheme in the Subset-Cover framework to be secure in the sense of Definition 2.

Theorem 14. *[NNL01, NNL02] Let $\mathcal{A}$ be a subset cover revocation algorithm where the key assignment satisfies the key indistinguishability property (Definition 1) and where E and F satisfy the above requirements. Then $\mathcal{A}$ is secure in the sense of Definition 2 with security parameter $\delta \leq \epsilon_a + 2h_{max}w(\epsilon_2 + 4w\epsilon_3)$, where w is the total number of subsets in the collection $\mathcal{S}$ of subsets for the scheme and h_{max} is the maximum size of a cover.*

Full resilience Full resilience of a broadcast encryption scheme is ensured if the collusion of all revoked users does not result in the correct decryption of the encrypted message. It is to be noted here that the key indistinguishability given in Definition 1 and the definition of security as given in Definition 2 together imply full resilience of the subset cover revocation algorithm. In other words, an adversary with the keys of all revoked users will not be able to distinguish the keys used for encrypting the message from random. As a result, the adversary will not be able to distinguish the encryption of a message from that of a random string with non-negligible probability.

Key indistinguishability for NNL-SD scheme Key indistinguishability of the NNL-SD scheme was not argued explicitly in [NNL01, NNL02]. Rather it was implied that this follows from the security property of the pseudo-random generator used to generate the keys for the subsets. We briefly highlight the PRG property required in [NNL01, NNL02].

Recall from Section 2.1 that every internal node i of $\mathcal{T}^0$ is assigned an independent and uniform random seed L_i. In the description of the key assignment technique of [NNL01, NNL02], a descendant node j of i, gets a seed $L_{i,j}$ derived from L_i using a cryptographic pseudo-random generator (PRG) $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$. The output of $G(seed)$ is divided into three equal parts $G_L(seed)$, $G_M(seed)$ and $G_R(seed)$, each of the same length as the input *seed* as defined in Section 2.1.

Recall that the users of the NNL-SD scheme are provided derived seeds $L_{i,j}$ as part of their secret information, as described in Section 2.1. The key associated to a subset $S_{i,j}$ is $G_M(L_{i,j})$. An adversary that corrupts a user u, has all its secret information I_u. Let us assume that the adversary has I_u for all users in $\mathcal{N} \setminus S_{i,j}$. The users in $\mathcal{N} \setminus S_{i,j}$ are either in $\mathcal{T}^0 \setminus \mathcal{T}^i$ or in $\mathcal{T}^j$. All seeds assigned to the users that are in $\mathcal{T}^0 \setminus \mathcal{T}^i$ are independent of the random seed L_i and hence of any other seed derived from it. Hence, the seeds assigned to the users in $\mathcal{T}^0 \setminus \mathcal{T}^i$ provide no information to the adversary about the key for $S_{i,j}$. This holds information theoretically and does not rely on the computational assumption on the PRG G.

A user in the subtree $\mathcal{T}^j$ gets seeds derived from the seed $L_{i,j}$ but, not $L_{i,j}$ itself. The PRG-security of G is used to argue that these seeds do not provide any information to the adversary. The key for the subset $S_{i,j}$

is $G_M(L_{i,j})$ whereas the users in $\mathcal{T}^j$ get either $G_L(L_{i,j})$, or $G_R(L_{i,j})$ or seeds derived from these two quantities using further applications of G_L or G_R.

The definition of a secure PRG says that it passes all polynomial time statistical tests. An equivalent formulation (modulo a tightness factor) is that a secure PRG passes the next bit test, i.e., given a segment of the output of the PRG, an adversary is unable to guess the next bit with probability significantly away from half. An easy extension of this notion is that given a segment of the output of a secure PRG, an adversary does not have significant advantage in guessing the bits of a disjoint segment of the output.

Applying this notion in the current context, we see that $G_M(L_{i,j})$, $G_L(L_{i,j})$ and $G_R(L_{i,j})$ are three disjoint segments of the outputs of G. So, for a secure PRG G, an adversary holding either or both of $G_L(L_{i,j})$ and $G_R(L_{i,j})$ does not have significant advantage in guessing $G_M(L_{i,j})$. As a result, holding I_u for all users u in $\mathcal{T}^j$ does not provide the adversary with significant advantage in learning the key $G_M(L_{i,j})$ associated to $S_{i,j}$. This argument (which is implicit in [NNL01, NNL02]) establishes the key indistinguishability of the NNL-SD scheme.

Key indistinguishability for the k-ary tree scheme The k-ary tree SD scheme is instantiated for fixed values of the parameters k and n. The underlying k-ary tree structure $\mathcal{T}^0$ is thus fixed with the users at the leaves. As described in Section 3.1, the subsets in $\mathcal{S}$ are of the form $S_{i,J}$ containing all leaves of $\mathcal{T}^i \setminus \cup_{j' \in J} \mathcal{T}^{j'}$ where J is a proper non-empty subset of the child nodes of a descendant node j of i in $\mathcal{T}^0$.

We recollect from Section 6.3 that in addition to the tree structure $\mathcal{T}^0$, an auxiliary tree structure $T^{(k)}$ is used for key assignment. An internal node j in $\mathcal{T}^0$ has k children nodes $kj+1, \ldots, kj+k$. There are $2^k - 2$ non-empty proper subsets J of these child nodes. The tree $T^{(k)}$ is such that each leaf node in it uniquely represents such a subset. Out of these, exactly k subsets are singleton sets and represent the k children of the internal node j in $\mathcal{T}^0$.

Two hash functions $F : [1,k] \times \{0,1\}^m \to \{0,1\}^m$ and $H : \{0,1,2\} \times \{0,1\}^m \to \{0,1\}^m$ are used to assign keys as explained in Section 6.3. The assignment of seeds starts with each internal node i of the underlying k-ary tree $\mathcal{T}^0$ being assigned an independent and uniform random seed L_i. Consider a subset $S_{i,J}$. To this subset is assigned a seed which is derived from L_i using F and H and the auxiliary tree $T^{(k)}$ as explained in Section 6.3. Denote this seed by $L_{i,J}$. The key for the subset $S_{i,J}$ is then $H_2(L_{i,J})$.

Suppose that the adversary has obtained the secret information for all users in $\mathcal{N} \setminus S_{i,J}$. As in the case of NNL-SD, we argue that this does not provide information about the key for $S_{i,J}$ to the adversary. The users in $\mathcal{N} \setminus S_{i,J}$ are either in $\mathcal{T}^0 \setminus \mathcal{T}^i$ or in $\cup_{j' \in J} \mathcal{T}^{j'}$. As earlier, the seeds assigned to users in $\mathcal{T}^0 \setminus \mathcal{T}^i$ are independent of L_i. Hence, an adversary with the seeds assigned to all users in $\mathcal{T}^0 \setminus \mathcal{T}^i$ has no information about the key associated to $S_{i,J}$. This property holds without any assumption on the security of H or F.

Consider now the secret information available to the users in the tree $\mathcal{T}^{j'}$ for any j' in J. Let j be the parent of the sibling nodes in J. The seed derived from L_i which is assigned to j using F and H is $L_{i,\{j\}}$. The seed $L_{i,J}$ is obtained by once applying F and then applying a sequence of H_1 and H_2 as determined by the position of J in the auxiliary tree $T^{(k)}$. By construction, none of the seeds in the path from the root of $T^{(k)}$ to the leaf node representing J is made available to any of the users in $\mathcal{T}^{j'}$. The information that is given to the users in $\mathcal{T}^{j'}$ is one of the following two forms:

1. Suppose F_b is applied to $L_{i,\{j\}}$ as the first step in the generation of $L_{i,J}$. Then $F_{b'}(L_{i,\{j\}})$ for $b \neq b'$ is possibly given out.

2. Suppose at some intermediate node in $T^{(k)}$, H_i ($i = 0$ or 1) is applied to a string x. Then $H_{1-i}(x)$ is possibly given out.

The key indistinguishability property follows if we can argue that F and H satisfy the following properties: Obtaining $F_b(z)$ does not reveal information about $F_{b'}(z)$; and obtaining $H_i(x)$ does not reveal information about $H_{1-i}(x)$ for $i = 0, 1$.

The condition on H is exactly the condition required for the PRG G in the NNL-SD scheme. So, we may consider the three functions H_0, H_1 and H_2 to be obtained by trifurcating the output of an m-bit to $3m$-bit PRG. Similarly, we may consider the functions $F_1, \ldots, F_k$ to be obtained by k-partitioning the output of an m-bit to km-bit PRG. With this instantiation, the security of the two PRGs provide the required conditions for key indistinguishability exactly as in the case of the NNL-SD scheme.

10 Conclusion

The most popular BE scheme is the NNL-SD scheme [NNL01, NNL02] that is defined on a binary tree structure. We present a generalization of the scheme which works with a k-ary tree for any $k \geq 2$. As a result, our work subsumes the NNL-SD scheme. We present detailed analysis of the user storage and the header length, the two important efficiency parameters of a BE scheme. This shows that if the number of revoked users is of the order of the number of total users, then using a k greater than 2 results in lower communication overhead at the cost of increased user storage. For applications where the increase in user storage can be tolerated, our work provides a wider variety of trade-off options between user storage and bandwidth.

Acknowledgment We would like to thank Subhabrata Samajder of the Indian Statistical Institute, Kolkata for his suggestion regarding cyclotomic cosets that was useful in reducing the user storage of our scheme. We also thank an anonymous reviewer for comments which helped to improve the paper.

References

[AAC] AACS. Advanced Access Content System, `http://www.aacsla.com`.

[AK08] Per Austrin and Gunnar Kreitz. Lower bounds for subset cover based broadcast encryption. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2008.

[Asa02] Tomoyuki Asano. A revocation scheme with minimal storage at receivers. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2002.

[Ber91] Shimshon Berkovits. How to broadcast a secret. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 535–541. Springer, 1991.

[BF87] Jon Bentley and Bob Floyd. Programming pearls: a sample of brilliance. *Commun. ACM*, 30(9):754–757, September 1987.

[BS13] Sanjay Bhattacherjee and Palash Sarkar. Complete tree subset difference broadcast encryption scheme and its analysis. *Des. Codes Cryptography*, 66(1-3):335–362, 2013.

[BS14] Sanjay Bhattacherjee and Palash Sarkar. Concrete analysis and trade-offs for the (complete tree) layered subset difference broadcast encryption scheme. *IEEE Trans. Computers*, 63(7):1709–1722, 2014.

[EOPR08] Christopher Eagle, Mohamed Omar, Daniel Panario, and Bruce Richmond. Distribution of the number of encryptions in revocation schemes for stateless receivers. In Uwe Roesler, Jan Spitzmann, and Marie-Christine Ceulemans, editors, *Fifth Colloquium on Mathematics and Computer Science*, volume AI of *DMTCS Proceedings*, pages 195–206. Discrete Mathematics and Theoretical Computer Science, 2008.

[FKTS08] K. Fukushima, S. Kiyomoto, T. Tanaka, and K. Sakurai. Ternary subset difference method and its quantitative analysis. In *The 9th International Workshop on Information Security Applications (WISA2008)*, volume 5379, pages 225–239. LNCS, 2008.

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient tree-based revocation in groups of low-state devices. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004.

[HS02] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002.

[JHC+05] Nam-Su Jho, Jung Yeon Hwang, Jung Hee Cheon, Myung-Hwan Kim, Dong Hoon Lee, and Eun Sun Yoo. One-way chain based broadcast encryption schemes. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2005.

[LS98] Michael Luby and Jessica Staddon. Combinatorial bounds for broadcast encryption. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 1998.

[MMW09] Thomas Martin, Keith M. Martin, and Peter R. Wild. Establishing the broadcast efficiency of the subset difference revocation scheme. *Des. Codes Cryptography*, 51(3):315–334, 2009.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

[NNL02] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. *Electronic Colloquium on Computational Complexity (ECCC)*, (043), 2002.

[PB06] E. C. Park and Ian F. Blake. On the mean number of encryptions for tree-based broadcast encryption schemes. *J. Discrete Algorithms*, 4(2):215–238, 2006.